Java学习群

72030155

进群可以免费获取视频教程以及每日免费听老师讲课

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

目　录

第一部分　Java概述 …… (1)
第1章　Java能做什么 …… (3)
1.1　Java应用程序的几种类型 …… (4)
1.2　了解Java语言 …… (5)
1.3　Java开发工具 …… (5)
1.4　Java小应用程序 …… (5)
1.5　Java图形用户界面应用程序 …… (9)
1.6　Java命令行应用程序 …… (11)
1.7　Java是客户/服务器方式的 …… (11)
1.8　如何跟上形势 …… (12)
第2章　Java设计 …… (13)
2.1　解释Java …… (14)
2.2　Java是面向对象的 …… (15)
2.3　Java虚拟机 …… (15)
2.3.1　Java源代码 …… (16)
2.3.2　Java堆栈 …… (16)
2.3.3　Java寄存器 …… (16)
2.3.4　无用单元收集堆 …… (17)
2.3.5　Java的方法区 …… (17)
2.4　安全性和Java虚拟机 …… (17)
2.4.1　可执行内容与安全性 …… (17)
2.4.2　Java实现安全性的办法 …… (19)
2.4.3　Java语言级的安全性 …… (20)
2.4.4　编译过的Java代码的安全性 …… (21)
2.4.5　Java系统在运行时刻的安全性 … (21)
2.4.6　可执行代码的安全性 …… (22)
2.4.7　开放问题 …… (24)
2.5　Java API …… (25)
2.5.1　Java核心API …… (25)
2.5.2　JavaBean API …… (27)
2.5.3　JDK 1.2新增加的内容 …… (28)
2.5.4　Java嵌入式API …… (29)
第3章　安装并启动JDK …… (30)
3.1　为什么需要使用Sun公司的Java开发工具来编写Java程序 …… (31)
3.2　进一步了解Java既是编译的又是解释的 …… (31)
3.3　获取和安装Sun公司的JDK …… (32)
3.3.1　从光盘安装在Windows 95和Windows NT上的JDK …… (33)
3.3.2　从光盘安装在x86和SPARC Solaris操作系统上的JDK …… (33)
3.3.3　下载JDK …… (34)
3.4　安装下载的JDK …… (35)
3.4.1　在Solaris x86平台和SPARC平台上安装 …… (35)
3.4.2　在Windows操作系统上安装 …… (36)
3.4.3　在Macintosh上安装 …… (37)
3.5　测试Java编译器和JVM(Java虚拟机) …… (38)
3.5.1　创建新的Java工程 …… (38)
3.5.2　运行Unix或Windows的Java应用程序 …… (38)
3.5.3　在Macintosh上运行Java应用程序 …… (39)
3.6　在Windows 3.1平台上安装IBM的小应用程序开发工具 …… (39)
第4章　JDK工具 …… (42)
4.1　JDK工具参考 …… (43)
4.2　AppletViewer(小应用程序浏览器) … (43)
4.3　java,Java解释器 …… (44)
4.4　javac,Java编译器 …… (45)
4.5　javap,Java反编译器 …… (46)
4.6　Javah C头文件和承接模块文件的创建器 …… (46)
4.7　javadoc工具(文档生成器) …… (47)
4.8　jdb,Java调试器 …… (48)
4.9　CLASSPATH环境变量 …… (48)
4.10　Macintosh问题 …… (49)
4.10.1　Macintosh平台上的小应用程序浏览器 …… (49)
4.10.2　Java Runner，Macintosh平台上的Java解释器 …… (50)
4.11　Java编译器 …… (51)
4.12　JavaH:C头文件产生器 …… (51)
第二部分　Java语言 …… (53)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 5 章 面向对象编程 …… (55)
5.1 面向对象编程:新的思维方式 …… (56)
5.2 简短的编程历史 …… (56)
5.2.1 过程语言 …… (56)
5.2.2 结构化程序设计 …… (57)
5.2.3 面向对象编程 …… (57)
5.3 对象 …… (57)
5.3.1 传统的程序设计 …… (58)
5.3.2 OOP 方法 …… (58)
5.3.3 通过继承扩充对象 …… (59)
5.4 多实体对象 …… (61)
5.5 组织代码 …… (61)
5.6 对象和对象如何与 Java 类联系 …… (61)
5.7 建立层次:一种面向对象设计的方法… (61)
5.7.1 把代码分成最小的实体 …… (62)
5.7.2 在实体间寻找共同点 …… (62)
5.7.3 在实体之间寻找差异 …… (62)
5.7.4 在实体间寻找最大的共同点 …… (62)
5.7.5 把剩下的共同对象放在一起并重复 …… (62)
5.7.6 利用对象扩充程序 …… (63)
5.8 Java 并不是完全面向对象的语言 …… (63)
第 6 章 Hello World:用户的第一个 Java 程序 …… (64)
6.1 HelloWorld 应用程序 …… (65)
6.1.1 创建文件 …… (65)
6.1.2 编译代码 …… (65)
6.1.3 运行程序 …… (65)
6.2 理解 HelloWorld …… (66)
6.2.1 声明类 …… (66)
6.2.2 Main 方法 …… (66)
6.2.3 写屏 …… (67)
6.2.4 System.out 和 system.in …… (67)
6.3 作为小应用程序的 HelloWorld 运行在 Netscape 中 …… (70)
6.3.1 新的源代码和编译新的源代码 … (70)
6.3.2 创建 HTML 文件 …… (71)
6.3.3 在小应用程序浏览器中运行程序 …… (71)
6.3.4 在 Netscape Navigator 中运行 HelloWold …… (72)
6.3.5 理解源代码 …… (72)
6.3.6 小应用程序简短的生命周期 …… (73)
6.4 关键字 …… (74)
6.5 API …… (75)
第 7 章 数据类型及其它标记 …… (76)
7.1 Java 的两种数据类型 …… (77)
7.2 了解布尔变量 …… (78)
7.2.1 声明变量 …… (78)
7.2.2 标识符:变量命名 …… (78)
7.2.3 修改布尔变量 …… (79)
7.3 各种整数类型 …… (80)
7.3.1 整数类型值的限制 …… (80)
7.3.2 创建整数变量 …… (81)
7.3.3 整数运算 …… (81)
7.4 运算符 …… (81)
7.4.1 算术运算符 …… (82)
7.4.2 赋值运算符 …… (82)
7.4.3 增量/减量运算符 …… (83)
7.5 字符变量 …… (83)
7.6 浮点变量 …… (84)
7.7 数组 …… (85)
7.8 空白 …… (86)
7.9 注释 …… (87)
7.9.1 传统的注释 …… (87)
7.9.2 C++ 风格的注释 …… (87)
7.9.3 javadoc 注释 …… (88)
7.10 源程序中的符号:赋值 …… (88)
7.10.1 代表整数格式的字符 …… (89)
7.10.2 代表字符格式的字符 …… (90)
7.10.3 代表浮点格式的字符 …… (91)
7.10.4 代表串格式的字符 …… (91)
7.11 创建和撤销对象 …… (92)
7.11.1 使用 new 操作符创建对象 …… (92)
第 8 章 方法 …… (94)
8.1 方法的两个部分 …… (95)
8.1.1 声明 …… (95)
8.2 块和语句 …… (103)
8.2.1 语句加标志 …… (104)
8.2.2 范围 …… (104)
8.3 分隔符 …… (105)
第 9 章 使用表达式 …… (106)
9.1 什么是表达式 …… (107)
9.2 如何计算表达式 …… (107)
9.2.1 运算符关联 …… (107)
9.2.2 Java 运算符的优先顺序 …… (108)
9.2.3 小结——运算符表 …… (109)
9.2.4 计算顺序 …… (109)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

9.3 Java 表达式与 C 语言表达式的差异 …… (110)
9.4 位运算符 …… (110)
9.5 移位运算符 …… (112)
9.6 类型转换 …… (112)
9.6.1 隐式类型转换 …… (112)
9.6.2 强制运算符 …… (113)
9.6.3 强制并转换整数 …… (114)
9.6.4 强制并转换字符 …… (114)
9.6.5 强制并转换布尔值 …… (114)
9.7 字符串加法 …… (114)
第 10 章 控制流 …… (115)
10.1 控制执行 …… (116)
10.2 布尔运算的 true 运算符和 false 运算符 …… (116)
10.2.1 关系运算符 …… (116)
10.2.2 等于运算符 …… (118)
10.3 逻辑表达式 …… (119)
10.3.1 条件与运算符和条件或运算符 … (119)
10.3.2 一元逻辑运算符 …… (120)
10.4 条件运算符 …… (120)
10.5 控制流中的布尔类型 …… (121)
10.6 控制流语句 …… (121)
10.6.1 if 语句 …… (121)
10.6.2 if-else 语句 …… (122)
10.7 循环语句 …… (123)
10.7.1 While 语句 …… (123)
10.7.2 do 语句 …… (124)
10.7.3 for 语句 …… (124)
10.7.4 switch 语句 …… (125)
10.8 跳转语句 …… (126)
10.8.1 break 语句 …… (126)
10.8.2 continue 语句 …… (126)
10.8.3 return 语句 …… (127)
第 11 章 类 …… (128)
11.1 什么是类 …… (129)
11.2 为什么使用类 …… (129)
11.3 Java 中的类 …… (130)
11.4 声明类 …… (133)
11.4.1 访问说明符 …… (133)
11.4.2 类名 …… (134)
11.4.3 超类——扩展另一个类 …… (134)
11.5 构造器 …… (135)
11.5.1 覆盖方法 …… (136)
11.6 创建类的实例 …… (137)
11.7 引用类的部分 …… (138)
11.7.1 特殊变量 this …… (140)
11.7.2 特殊变量 super …… (141)
11.8 变量 …… (142)
11.8.1 变量修饰符 …… (146)
11.8.2 使用方法来提供受保护的访问 … (148)
11.8.3 使用 findlize()方法 …… (149)
11.9 嵌入类 …… (151)
11.9.1 什么是嵌入类 …… (151)
11.9.2 使用嵌入类创建程序 …… (151)
11.9.3 同步嵌入类 …… (153)
11.9.4 嵌入类是如何工作的 …… (154)
11.9.5 为什么使用嵌入类 …… (154)
11.10 包 …… (154)
11.11 引入包中的类 …… (155)
11.12 引入整个包 …… (155)
11.13 不通过引入使用类 …… (156)
11.14 使用包来组织代码 …… (158)
11.15 所有 java.lang 类的隐式引入 …… (158)
第 12 章 接口 …… (159)
12.1 什么是接口 …… (160)
12.2 创建接口 …… (160)
12.2.1 声明 …… (162)
12.2.2 接口体 …… (163)
12.3 使用接口 …… (166)
12.3.1 覆盖方法 …… (166)
12.3.2 修饰符 …… (166)
12.3.3 参数列表 …… (167)
12.3.4 体 …… (168)
12.4 在其它类中使用接口 …… (168)
12.4.1 使用接口的域 …… (168)
12.4.2 把接口当作类型使用 …… (168)
12.5 异常 …… (171)
第 13 章 线程 …… (173)
13.1 什么是线程 …… (174)
13.2 为什么使用线程 …… (174)
13.3 如何使类可线程化 …… (174)
13.3.1 扩展 Thread 类 …… (174)
13.3.2 使用 Runnable 接口 …… (175)
13.4 巧妙的线程竞赛 …… (175)
13.5 理解 GreatRace …… (178)
13.6 线程处理 …… (180)
13.7 测试巧妙的线程竞赛 …… (181)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

13.8 改变优先级 …… (182)
13.9 关于线程的优先级、Netscape 和 Windows …… (183)
13.10 同步 …… (185)
13.11 哲学家进餐问题 …… (186)
13.12 改变线程的运行状态 …… (186)
13.13 获得正在运行的线程的数量 …… (188)
13.14 查找所有正在运行的线程 …… (188)
13.15 超级属性 …… (190)
第 14 章 编写小应用程序 …… (192)
14.1 Java 的孩子 …… (193)
14.2 小应用程序和 HTML 语言 …… (193)
14.3 在 HTML 页中包含 Java 小应用程序 …… (194)
14.3.1 包括替换信息 …… (195)
14.3.2 附加的 < APPLET > 属性 …… (197)
14.4 使用 Java 插件 …… (198)
14.4.1 在 Internet Explorer 上使用 Java 插件 …… (199)
14.4.2 在 Netscape 中使用 Java 插件 …… (199)
14.4.3 为所有的浏览器设置 HTML 文本 …… (200)
14.5 开始开发 Java 小应用程序 …… (201)
14.5.1 理解 HelloWorld——构造小应用程序 …… (203)
14.6 探究小应用程序的生命周期 …… (203)
14.6.1 编译 InitStartStop 小应用程序 …… (206)
14.6.2 理解 InitStartStop 小应用程序 …… (207)
14.6.3 Java 动画小应用程序 …… (209)
14.7 使用控件的小应用程序 …… (213)
14.7.1 理解 InternetApplet 小应用程序 …… (218)
14.7.2 探究 init( )方法 …… (218)
14.7.3 ActionCommand 和 ActionListeners …… (219)
14.7.4 actionPeformed( )方法 …… (219)
14.7.5 探究 paint( )方法 …… (220)
第 15 章 高级小应用程序代码 …… (222)
15.1 使用 < PARAM > 标记 …… (223)
15.1.1 理解 starPainter 的源代码 …… (224)
15.1.2 使用 getSize( )方法 …… (225)
15.2 在小应用程序中增加图像 …… (227)
15.3 在小应用程序中增加声音 …… (229)
15.4 使用小应用程序控制浏览器 …… (230)
15.4.1 修改状态信息 …… (231)
15.4.2 修改浏览器显示的页 …… (232)
15.5 小结 …… (233)
第 16 章 JAR 档案文件 …… (235)
16.1 为什么使用 JAR …… (236)
16.1.1 绑定 …… (236)
16.1.2 压缩 …… (236)
16.1.3 向下兼容性 …… (236)
16.1.4 可移植性 …… (236)
16.1.5 安全性 …… (237)
16.2 什么时候使用 JAR 档案 …… (237)
16.2.1 Jar 工具 …… (237)
16.2.2 < APPLET > 标志 …… (239)
16.2.3 兼容的浏览器 …… (241)
16.3 JAR 档案和安全性 …… (241)
16.3.1 声明文件 …… (242)
16.3.2 私有密钥、公共密钥和证书 …… (243)
16.3.3 密钥工具 …… (243)
16.3.4 jarsigner …… (244)
16.4 java.util.zip 包 …… (246)
16.4.1 类 …… (247)
16.4.2 有计划地读取 JAR 文件 …… (247)
16.5 JAR 文件格式 …… (249)
第 17 章 小应用程序和应用程序的比较 …… (250)
17.1 应用程序探究 …… (251)
17.2 应用程序的优点 …… (251)
17.3 开发 Java 应用程序 …… (252)
17.3.1 HelloWorld 应用程序 …… (252)
17.3.2 为应用程序传递参数 …… (253)
17.3.3 避免空指针异常 …… (254)
17.3.4 由 main( )的静态特性带来的限制 …… (255)
17.4 把小应用程序转换为应用程序 …… (257)
17.4.1 为什么要把小应用程序转换为应用程序 …… (257)
17.4.2 把小应用程序代码修改成应用程序代码 …… (257)
17.4.3 解释其它小应用程序的特性 … (262)
17.4.4 缺省 …… (262)
17.4.5 加入 < param > 信息的第二种

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

办法 …… (264)
17.4.6 使用窗口来结束应用程序的工作 …… (265)
17.4.7 检查所有的小应用程序的方法 …… (269)
17.5 使用 Zip 文件格式将用户的应用程序打包 …… (270)
17.6 把应用程序转换成小应用程序 …… (270)
17.6.1 最简单的转换 …… (271)
17.6.2 处理命令行参数 …… (271)
17.6.3 维护应用程序的一个单一实例… (271)
17.6.4 在应用程序向小应用程序转换过程中更难的问题 …… (272)
第 18 章 管理应用程序 …… (273)
18.1 安装应用程序 …… (274)
18.1.1 从.class 文件中安装应用程序 …… (274)
18.1.2 结束安装 …… (275)
18.1.3 结束 UNIX 操作系统下的应用程序安装 …… (275)
18.1.4 结束 Windows 操作系统下的应用程序安装 …… (277)
18.1.5 从.jar 文件中安装应用程序 … (280)
18.2 在同一个系统中维护多个应用程序… (281)
第三部分 用户界面 …… (283)
第 19 章 java.awt:部件 …… (285)
19.1 使用 java.awt 构造图形用户界面 … (286)
19.2 按钮 …… (287)
19.2.1 创建按钮 …… (287)
19.2.2 使用按钮 …… (287)
19.3 按照 1.1 版本的事件模式使用按钮… (290)
19.4 标签 …… (291)
19.5 复选框和单选按钮 …… (292)
19.5.1 创建复选框 …… (292)
19.5.2 检查和设置复选框的状态 …… (293)
19.5.3 监听复选框中的变化 …… (293)
19.5.4 创建单选按钮 …… (295)
19.5.5 使用单选按钮 …… (296)
19.6 选择框 …… (299)
19.6.1 创建选择框 …… (299)
19.6.2 使用选择框 …… (300)
19.7 列表框 …… (301)
19.7.1 创建列表框 …… (301)
19.7.2 列表框特性 …… (301)
19.7.3 使用列表框 …… (303)
19.8 文本域和文本区 …… (307)
19.8.1 创建文本域 …… (307)
19.8.2 创建文本区 …… (308)
19.8.3 公共文本部件的特性 …… (308)
19.8.4 文本域特性 …… (309)
19.8.5 文本区特性 …… (309)
19.8.6 使用文本域和文本区 …… (310)
19.9 滚动条 …… (313)
19.9.1 创建滚动条 …… (313)
19.9.2 滚动条特性 …… (313)
19.9.3 使用滚动条 …… (314)
19.10 画布 …… (315)
19.11 部件的公共方法 …… (317)
19.11.1 部件显示方法 …… (317)
19.11.2 部件的位置和尺寸设置 …… (318)
19.11.3 部件布局和送显的方法 …… (320)
19.11.4 部件输入事件 …… (321)
第 20 章 深入研究异常和事件…… (323)
20.1 Java 的异常 …… (324)
20.1.1 抛出异常 …… (326)
20.1.2 联合方案 …… (327)
20.1.3 异常的类型 …… (328)
20.1.4 确定要处理的异常 …… (330)
20.1.5 捕捉运行时刻异常 …… (333)
20.1.6 处理异常 …… (334)
20.1.7 创建用户自己的异常类 …… (336)
20.2 Java 的 Error 类 …… (340)
20.3 Java 的事件 …… (340)
20.4 事件处理技术 …… (341)
20.4.1 事件监听器 …… (341)
20.5 键盘和鼠标事件 …… (345)
20.5.1 键盘事件 …… (345)
20.5.2 Java 2 中的修饰键 …… (348)
20.5.3 Java 2 中的鼠标事件 …… (349)
20.6 Java 1.0 的事件模式 …… (350)
20.6.1 Event 类 …… (350)
20.6.2 事件的起源 …… (353)
20.6.3 键盘 …… (355)
20.6.4 直接处理事件 …… (356)
20.6.5 覆盖 handleEvent( )方法 …… (356)
20.6.6 发送用户自己的事件 …… (358)
第 21 章 容器和布局管理器…… (362)
21.1 组织部件 …… (363)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

21.2 容器 …… (363)
21.3 布局管理器 …… (363)
21.4 再谈容器 …… (364)
21.5 容器基础 …… (364)
21.6 面板 …… (365)
21.7 框架 …… (366)
21.7.1 创建框架 …… (366)
21.7.2 框架特性 …… (367)
21.7.3 使用框架使用户的小应用程序单独运行 …… (368)
21.7.4 为框架增加菜单 …… (369)
21.7.5 使用菜单 …… (370)
21.7.6 弹出式菜单 …… (373)
21.8 对话框 …… (373)
21.8.1 创建对话框 …… (373)
21.8.2 可重用的 OK 对话框 …… (374)
21.9 滚动窗格 …… (378)
21.10 再谈布局管理器 …… (378)
21.10.1 流式布局 …… (379)
21.10.2 网格布局 …… (380)
21.10.3 方位布局 …… (381)
21.10.4 网格块布局 …… (382)
21.11 镶边 …… (386)
21.12 空布局管理器 …… (387)
21.13 从 Sun 公司的产品进一步扩展 …… (387)
第 22 章 图形 …… (388)
22.1 Java 图形 …… (389)
22.2 paint 方法、Update 方法和 repaint 方法 …… (389)
22.3 Graphics 类 …… (389)
22.3.1 坐标系统 …… (390)
22.3.2 画线 …… (390)
22.3.3 画矩形 …… (391)
22.3.4 画三维矩形 …… (391)
22.3.5 画圆角矩形 …… (393)
22.3.6 画圆和椭圆 …… (394)
22.3.7 画多边形 …… (395)
22.4 Polygon 类 …… (396)
22.5 画文本 …… (397)
22.5.1 Font 类 …… (399)
22.5.2 FontMetrics 类 …… (402)
22.6 画模式 …… (403)
22.7 绘制图像 …… (406)
22.8 MediaTracker 类 …… (408)
22.9 实用图形类 …… (411)
22.9.1 Point 类 …… (411)
22.9.2 Dimension 类 …… (412)
22.9.3 Rectangle 类 …… (412)
22.10 Color 类 …… (414)
22.11 裁剪 …… (416)
22.12 动画技术 …… (417)
22.13 打印 …… (419)
第 23 章 JFC——Java 基础类 …… (421)
23.1 Java 基础类 …… (422)
23.2 JFC:概览 …… (424)
23.2.1 设置 JFC …… (424)
23.3 Helloworld …… (424)
23.3.1 编译 HellooWorldJFC …… (425)
23.3.2 运行中的 HelloworldJFC …… (425)
23.3.3 理解 HelloWorldJFC …… (426)
23.3.4 窗格分层 …… (426)
23.4 改进 Helloworld …… (426)
23.4.1 JLabel …… (427)
23.4.2 增加图标 …… (428)
23.4.3 关闭窗口 …… (428)
23.5 使用 JFC 增加按钮 …… (429)
23.5.1 理解 ButtonExample …… (430)
23.5.2 设置快捷键和助记符 …… (430)
23.5.3 监听来自于按钮的动作 …… (430)
23.6 增加工具提示和图标 …… (431)
23.7 使用弹出式菜单 …… (432)
23.7.1 理解 PopupExample …… (434)
23.8 边 …… (435)
23.8.1 理解 BorerButtous …… (436)
23.8.2 更多的边 …… (436)
23.9 复选框和单选按钮 …… (437)
23.9.1 理解 CheckBoxPanel …… (439)
23.9.2 使用 ActionListeners 和设置事件 …… (439)
23.9.3 使用箱和箱式布局 …… (439)
23.10 使用 CheckBoxPanel 改变文本对齐方式 …… (440)
23.10.1 理解 CheckBoxPanel …… (442)
23.10.2 改变对齐方式 …… (442)
23.11 标签窗格 …… (442)
23.11.1 理解 JTabbedPane …… (443)
23.11.2 其它的 JTabbedPane 能力 …… (444)
23.12 滑杆 …… (444)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

23.12.1 理解 SliderExample …… (446)
23.12.2 设置刻度标志 …… (446)
23.12.3 捕捉滑杆的变化 …… (446)
23.13 进度条 …… (446)
23.13.1 理解 ProgressBarExample …… (449)
23.13.2 创建和控制进度条 …… (449)
23.13.3 ProgressThread …… (450)
23.13.4 RrogressMonitor …… (450)
23.13.5 ProgressMonitorInputStream ……
…… (453)
第 24 章 高级 JFC …… (456)
24.1 模型-视-控制——JFC 设计 …… (457)
24.1.1 比较 MVC 和 AWT 设计 …… (457)
24.1.2 这种模式的部件用在哪里 …… (458)
24.1.3 深入研究 MVC 部件是如何工作的 …… (458)
24.1.4 为什么要理解 MVC …… (459)
24.1.5 使用 JFC 的可插入的外观-和-感觉 …… (460)
24.1.6 修改外观-和-感觉 …… (461)
24.2 菜单和工具条 …… (462)
24.2.1 理解 MenuBar 示例 …… (465)
24.2.2 加速键 …… (466)
24.3 列表框和组合框 …… (467)
24.3.1 理解 ListCombExample …… (468)
24.3.2 列表视模型 …… (468)
24.4 使用表格 …… (470)
24.4.1 理解 TableExample …… (471)
24.4.2 表格模式 …… (471)
24.4.3 单元编辑器 …… (473)
24.5 树 …… (475)
24.5.1 树结点 …… (475)
24.5.2 树模型 …… (477)
24.5.3 理解新的 TreeExample …… (480)
24.5.4 removeSelctedNode …… (480)
24.5.5 AddTextFieldValue( ) …… (480)
24.6 使用 JEditor 显示 HTML 文件 …… (481)
24.6.1 理解 HTMLView …… (483)
24.6.2 监听超链事件 …… (483)
24.6.3 设置新页 …… (483)
24.6.4 JTextField …… (484)
24.7 创建图标 …… (484)
24.8 JFC 小应用程序 …… (487)
第 25 章 图像 …… (488)
25.1 在屏幕上绘制图像 …… (489)
25.2 生产者、消费者和观察者 …… (492)
25.3 图像过滤器 …… (494)
25.4 从内存拷贝图像 …… (496)
25.5 将图像拷贝到内存 …… (498)
25.6 颜色模式 …… (506)
25.6.1 DirectColorModel 类 …… (507)
25.6.2 IndexColorModel 类 …… (508)
25.6.3 RGBImageFilter 类 …… (508)
25.6.4 通过颜色循环产生动画 …… (513)
第 26 章 Java 二维图形 …… (520)
26.1 Graphics2D 对象 …… (521)
26.2 Java 二维应用编程接口中的坐标 … (521)
26.3 画图 …… (522)
26.3.1 画线 …… (522)
26.3.2 绘制矩形 …… (523)
26.3.3 绘制圆角矩形 …… (523)
26.3.4 绘制椭圆和圆 …… (523)
26.3.5 画弧 …… (523)
26.3.6 画曲线 …… (524)
26.3.7 绘制任意形状 …… (525)
26.4 不同的笔划 …… (525)
26.5 自定义填充 …… (527)
26.6 转换 …… (530)
26.7 绘制文本 …… (532)
26.7.1 有风格设置的字符串 …… (532)
26.7.2 文本布局 …… (533)
26.7.3 字符属性 …… (534)
26.8 绘制图像 …… (535)
26.8.1 缓冲区中的图像 …… (535)
26.8.2 将图像拷贝到 BufferedImage 中 …… (536)
26.8.3 过滤缓冲区里的图像 …… (536)
26.8.4 操作缓冲区图像 …… (542)
26.9 透明 …… (546)
26.10 裁剪 …… (547)
第四部分 输入输出 …… (551)
第 27 章 文件和流 …… (553)
27.1 什么是流 …… (554)
27.2 基本输入输出类 …… (555)
27.2.1 InputStream 类 …… (555)
27.2.2 OutputStream 类 …… (557)
27.2.3 System.in 对象和 System.out 对象 …… (557)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

27.2.4 PrintWriter 类 …… (558)
27.3 处理文件 …… (560)
27.3.1 文件安全 …… (560)
27.3.2 FileInputStream 类 …… (560)
27.3.3 使用 FileOutputStream 类 …… (561)
27.3.4 使用 File 类 …… (563)
27.3.5 RandomAccessFile 类 …… (567)
27.4 使用管道 …… (569)
27.4.1 PipedInputStream 类和 PipedOutputStream 类 …… (569)
27.4.2 PipeApp 应用程序 …… (570)
27.4.3 探究 main( )方法 …… (573)
27.4.4 探究 changeToY( )方法 …… (575)
27.4.5 探究 changeToZ( )方法 …… (576)
27.4.6 探究 YThread 类 …… (577)
27.4.7 过滤流 …… (578)
27.5 缓冲区里的流 …… (578)
27.6 数据流 …… (578)
27.6.1 DataInput 接口 …… (579)
27.6.2 DataOutput 接口 …… (579)
27.6.3 DataInputStream 类和 DataOutputStream 类 …… (580)
27.7 字节数组流 …… (580)
27.8 字符数组流 …… (581)
27.9 字节和字符之间的转换 …… (581)
27.10 StringBufferInputStream …… (582)
27.11 对象流 …… (582)
27.12 其它流 …… (585)
27.12.1 LineNumberReader 类 …… (585)
27.12.2 SequenceInputStream 类 …… (586)
27.12.3 PushbackInputStream 类 …… (586)
27.13 StreamTokenizer 类 …… (587)
第 28 章 使用字符串和文本 …… (590)
28.1 字符串简介 …… (591)
28.2 使用 String 类 …… (592)
28.2.1 获取关于 String 对象的信息 … (593)
28.2.2 比较字符串 …… (596)
28.2.3 字符串抽取 …… (599)
28.2.4 字符串操作 …… (602)
28.3 使用 StringBuffer 类 …… (603)
28.3.1 创建 StringBuffer 对象 …… (603)
28.3.2 获取关于 StringBuffer 对象的信息 …… (604)
28.3.3 StringBuffer 对象 …… (604)
28.3.4 操作 StringBuffer 对象 …… (604)
28.4 使用 StringToKenizer 类 …… (605)
28.5 字体处理 …… (608)
28.5.1 获取字体属性 …… (608)
28.5.2 获取字体度量 …… (610)
28.5.3 创建字体 …… (611)
28.5.4 使用字体 …… (611)
第 29 章 使用多国语言实现 …… (614)
29.1 为什么使用多国语言实现 …… (615)
29.2 什么是多国语言实现 …… (615)
29.3 Java 对多国语言实现的支持 …… (615)
29.3.1 Locale 类 …… (616)
29.3.2 将场所敏感数据打包 …… (618)
29.3.3 Java.util 包中的其它与多国语言实现有关的部分 …… (621)
29.4 多国语言实现的输入和输出(I/O) …… (621)
29.4.1 字符集转换器 …… (622)
29.4.2 Reader 和 Wtrter …… (623)
29.5 新的 java.text 包 …… (624)
29.6 示例:InternationalTest …… (625)
第 30 章 通信和网络 …… (629)
30.1 TCP/IP 概述 …… (630)
30.1.1 OSI 参考模型 …… (630)
30.1.2 TCP/IP 网络模型 …… (630)
30.2 TCP/IP 协议 …… (633)
30.2.1 Internet 协议(IP) …… (633)
30.2.2 传输控制协议(TCP) …… (633)
30.2.3 用户数据报协议(UDP) …… (634)
30.3 统一资源定位器(URL) …… (634)
30.3.1 URL 语法 …… (634)
30.3.2 一般 URL 格式 …… (634)
30.4 Java 和 URL …… (635)
30.4.1 URL 类 …… (635)
30.4.2 连接到 URL 上 …… (636)
30.4.3 以 HTTP 为中心的类 …… (637)
30.4.4 示例:定制 AltaVista 搜索引擎 … (637)
第 31 章 TCP 套接字 …… (644)
31.1 TCP 套接字基础 …… (645)
31.1.1 什么是套接字 …… (645)
31.1.2 Java TCP 套接字类 …… (647)
31.1.3 定制套接字行为 …… (649)
31.2 创建 TCP 客户/服务器应用程序 … (650)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

31.2.1 设计一个应用程序协议 …… (650)
31.2.2 开发股票客户程序 …… (651)
31.2.3 开发股票报价服务器程序 …… (656)
31.2.4 运行客户和服务器应用程序 …… (663)
第32章 UDP套接字 …… (664)
32.1 UDP信息概览 …… (665)
32.1.1 UDP套接字的特性 …… (665)
32.1.2 Java UDP类 …… (666)
32.2 创建UDP服务器程序 …… (667)
32.2.1 启动服务器 …… (671)
32.2.2 StartServing()方法处理请求 …… (671)
32.2.3 getTimeBuffer()方法创建字节数组 …… (671)
32.2.4 运行时间服务器程序 …… (671)
32.3 创建UDP客户程序 …… (671)
32.3.1 启动TimeCompare …… (676)
32.3.2 getTimes()方法:TimeCompare的执行路径 …… (677)
32.3.3 printTimes()方法:显示比较结果 …… (677)
32.3.4 运行应用程序 …… (677)
32.4 使用IP多点发送 …… (678)
32.4.1 Java的多点发送 …… (679)
32.4.2 多点发送应用程序 …… (680)
第33章 java.net …… (686)
33.1 URL类 …… (687)
33.1.1 获取URL内容 …… (688)
33.1.2 获取URL信息 …… (688)
33.2 URLConnection类 …… (689)
33.3 HTTPURLConnection类 …… (691)
33.4 URLEncoder类 …… (691)
33.5 URLDecoder类 …… (692)
33.6 URLStreamHandler类 …… (692)
33.7 ContentHandler类 …… (693)
33.8 Socket类 …… (693)
33.8.1 发送和接收Socket数据 …… (694)
33.8.2 获取Socket信息 …… (695)
33.8.3 设置Socket选项 …… (695)
33.8.4 关闭Socket连接 …… (696)
33.8.5 等待到来的数据 …… (696)
33.8.6 一个简单的Socket客户程序 …… (698)
33.9 ServerSocket类 …… (701)
33.9.1 接收进入的套接字连接 …… (701)
33.9.2 获得服务器套接字地址 …… (702)
33.9.3 编写一个服务器程序 …… (702)
33.10 InetAddress类 …… (706)
33.10.1 转换名字为地址 …… (706)
33.10.2 检验InetAddress …… (707)
33.10.3 获得一个Java小程序的起始地址 …… (707)
33.11 DatagramSocket类 …… (707)
33.12 DatagramPacket类 …… (709)
33.12.1 广播数据报 …… (709)
33.12.2 一个简单的数据报服务器程序 …… (710)
33.13 多点发送套接字 …… (712)
第34章 Java安全性深析 …… (714)
34.1 Java安全性的必要性 …… (715)
34.2 Java安全性框架 …… (716)
34.2.1 第一部分:语言所提供的安全措施 …… (716)
34.2.2 第二部分:Java编译器 …… (717)
34.2.3 第三部分:检验器 …… (717)
34.2.4 第四部分:类装载器 …… (718)
34.2.5 第五部分:建立安全性策略 …… (718)
34.2.6 总结 …… (718)
34.3 Java小程序的限制 …… (719)
34.3.1 Java小程序与应用程序 …… (719)
34.3.2 SecurityManager类 …… (720)
34.3.3 Java浏览器的安全性策略 …… (720)
34.4 Java安全性问题 …… (723)
34.4.1 已知的缺陷 …… (724)
34.4.2 拒绝服务攻击 …… (725)
34.5 Java安全性API:扩展Java小程序边界 …… (725)
34.5.1 均衡密码技术(Symmetric Cryptography) …… (726)
34.5.2 公共密钥加密 …… (727)
34.5.3 资格授权 …… (728)
34.5.4 实现了什么 …… (729)
34.5.5 密钥管理 …… (729)
34.5.6 为一个JAR文件数字签名 …… (730)
34.5.7 定义策略 …… (731)
34.5.8 运行Java小程序 …… (731)
34.6 安全性API …… (731)
34.6.1 公共密钥和私人密钥类 …… (731)
34.6.2 Signature类 …… (732)
34.6.3 身份及签名者 …… (732)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

34.6.4 资格证 …… (733)
34.6.5 IdentityScope 类 …… (733)
第 35 章 对象系列化 …… (736)
35.1 什么是对象系列化 …… (737)
35.1.1 对象系列化如何运作 …… (740)
35.1.2 使用对象引用处理对象 …… (740)
35.2 对象系列化示例 …… (740)
35.2.1 一个写 Date 类的应用程序 …… (740)
35.2.2 在 JDK1.02 下运行 DateWrite … (742)
35.2.3 编译和运行 DateWrite …… (742)
35.2.4 一个读入 Date 类的简单应用程序 …… (742)
35.2.5 编译和运行 DateRead …… (744)
35.2.6 使用 Java 小程序读入 Date 类 …… (744)
35.3 写和读自己的对象 …… (745)
35.4 定制对象系列化 …… (747)
第 36 章 远程方法调用 …… (750)
36.1 什么是远程方法调用 …… (751)
36.1.1 创建远程对象 …… (751)
36.1.2 一个 RMI 应用程序样本 …… (751)
36.1.3 创建远程接口 …… (752)
36.1.4 创建实现类 …… (752)
36.1.5 编译 RemoteServer …… (754)
36.1.6 创建主干(Stub)…… (754)
36.1.7 创建客户程序 …… (755)
36.1.8 启动注册并运行代码 …… (756)
36.1.9 绑定 RemoteObject 到注册 …… (757)
36.1.10 运行客户程序 …… (757)
36.2 创建 Java 小程序客户程序 …… (757)
36.3 创建定制套接字 …… (758)
36.3.1 创建一个定制套接字 …… (759)
36.3.2 创建定制的 RMISocketFactory… (762)
36.3.3 在应用程序中指定套接字工厂… (763)
36.4 使用激活模型 …… (765)
36.4.1 创建可激活对象 …… (766)
第 37 章 管理 API …… (769)
37.1 JMAPI 组件 …… (770)
37.2 JMAPI 小程序 …… (770)
37.2.1 JmapiHome.html …… (770)
37.2.2 MOContentManagerApplet.html… (771)
37.2.3 MOPropertyBookApplet.html …… (771)
37.2.4 访问管理对象 …… (771)
37.3 创建管理对象 …… (772)
37.3.1 管理对象中定义属性 …… (772)
37.3.2 管理对象中定义方法 …… (772)
37.3.3 编译管理对象 …… (773)
37.3.4 导入管理对象 …… (773)
37.4 Admin View Module …… (773)
37.4.1 内容管理器和可选择对象 …… (773)
37.4.2 图标 …… (777)
37.4.3 属性册 …… (780)
37.4.4 任务页 …… (781)
37.4.5 对话框 …… (784)
37.4.6 自验证字段 …… (789)
37.4.7 新布局管理器 …… (791)
第五部分 数据库 …… (793)
第 38 章 数据库介绍 …… (795)
38.1 ODBC 和 JDBC …… (796)
38.2 关系数据库概念 …… (796)
38.2.1 SQL …… (798)
38.2.2 连接 …… (798)
38.3 ODBC 技术概述…… (801)
38.3.1 ODBC 一致层…… (803)
38.3.2 ODBC 函数和命令集…… (804)
38.4 高级客户/服务器概念 …… (805)
38.4.1 客户/服务器系统层 …… (805)
38.4.2 事务 …… (806)
38.4.3 数据光标 …… (807)
38.4.4 复制 …… (809)
第 39 章 JDBC:Java 数据库连接 …… (810)
39.1 JDBC 概述 …… (811)
39.1.1 JDBC 是如何工作的 …… (811)
39.1.2 安全模型 …… (813)
39.1.3 JDBC-ODBC 桥接器 …… (813)
39.2 JDBC 实现 …… (813)
39.2.1 JDBC 类——概述 …… (814)
39.2.2 JDBC 应用程序剖析 …… (816)
39.2.3 JDBC 实例 …… (816)
39.3 Connection 类 …… (819)
39.4 元数据函数 …… (820)
39.4.1 DatabaseMetaData …… (821)
39.4.2 ResultSetMetaData …… (825)
39.5 SQLException 类 …… (826)
39.6 SQLWarning 类 …… (827)
第 40 章 JDBC 考察 …… (828)
40.1 语句 …… (829)
40.1.1 Statement …… (830)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

40.1.2 PreparedStatement …… (831)
40.1.3 CallableStatement …… (834)
40.2 ResultSet 处理检查结果 …… (836)
40.3 其它 JDBC 类 …… (838)
40.3.1 java.sql.Date …… (838)
40.3.2 java.sql.Time …… (838)
40.3.3 java.sql.Timestamp …… (839)
40.3.4 java.sql.Types …… (839)
40.3.5 java.sql.DataTruncation …… (840)
40.4 JDBC 的前景 …… (841)
第六部分 基于组件的开发 …… (843)
第 41 章 JavaBeans …… (845)
41.1 自包含的组件 …… (846)
41.2 组件模型中的重要概念 …… (846)
41.2.1 组件字段或属性 …… (846)
41.2.2 组件方法或函数 …… (847)
41.2.3 事件和相互通信 …… (847)
41.2.4 状态维持和存储 …… (847)
41.3 设计 JavaBean 的基础 …… (847)
41.3.1 指定 Bean 的属性 …… (848)
41.3.2 指定 Bean 所产生或响应的事件 …… (848)
41.3.3 属性、方法及事件的公开 …… (849)
41.3.4 初始化属性值和 Bean 的定制 …… (849)
41.4 创建和使用属性 …… (851)
41.4.1 单值属性 …… (851)
41.4.2 索引属性 …… (853)
41.4.3 绑定性属性 …… (854)
41.4.4 限制性属性 …… (855)
41.5 使用事件与其它组件通信 …… (856)
41.5.1 多点发送事件 …… (856)
41.5.2 单点发送事件 …… (857)
41.5.3 事件适配器 …… (857)
41.6 内省:建立和使用 BeanInfo 类 …… (859)
41.7 定制:提供定制属性编辑器和 GUI 接口 …… (861)
41.7.1 属性编辑器和属性编辑器管理器 …… (861)
41.7.2 定制编辑器 …… (862)
41.7.3 在非 GUI 环境中提供另一种可选行为 …… (864)
41.8 企业级 JavaBeans …… (865)
41.8.1 服务器组件 …… (865)
41.8.2 添加组件的"随处运行"功能 …… (866)
41.8.3 划分应用程序 …… (866)
41.8.4 可重用性及集成 …… (866)
41.8.5 非可视组件 …… (866)
41.8.6 命名 …… (866)
第 42 章 JavaIDL:Java 的 CORBA 接口 …… (867)
42.1 什么是 CORBA …… (868)
42.2 Sun 公司的 IDL 到 Java 的映射 …… (870)
42.2.1 IDL 模块 …… (870)
42.2.2 IDL 常量 …… (871)
42.2.3 IDL 数据类型 …… (871)
42.2.4 枚举类型 …… (871)
42.2.5 结构 …… (872)
42.2.6 联合 …… (873)
42.2.7 序列与数组 …… (875)
42.2.8 异常 …… (875)
42.2.9 接口 …… (876)
42.2.10 属性 …… (876)
42.3 方法 …… (876)
42.4 创建基本的 CORBA 服务器 …… (877)
42.4.1 编译 IDL 定义 …… (878)
42.4.2 使用 IDL 的结构所定义的类 …… (878)
42.4.3 JavaIDL 框架 …… (880)
42.4.4 服务器初始化 …… (884)
42.5 使用 JavaIDL 创建 CORBA 客户程序 …… (885)
42.6 在 CORBA 中创建回叫 …… (888)
42.7 CORBA 与现有对象的包装 …… (888)
42.7.1 与 CORBA 所定义类型的映射 … (889)
42.7.2 创建远程方法包装 …… (890)
42.8 在小程序中使用 CORBA …… (891)
42.8.1 选择 CORBA 还是 RMI …… (891)
第 43 章 Java-COM 集成 …… (893)
43.1 一个重要的扩充 …… (894)
43.2 COM 概览 …… (894)
43.3 定义 COM 接口 …… (896)
43.4 编译 ODL 文件 …… (898)
43.5 生成 GUID …… (898)
43.6 使用 Java 创建 COM 对象 …… (898)
43.7 从 Visual Basic 调用 Java COM 对象 …… (901)
43.8 从 Excel 调用 Java 对象 …… (902)
43.9 从 Java 调用 COM 对象 …… (904)
第七部分 高级 Java …… (907)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 44 章 Java 媒体框架 …… (909)
44.1 什么是 Java 媒体框架 …… (910)
44.2 创建媒体播放器 …… (910)
44.2.1 预提取媒体 …… (911)
44.2.2 添加播放器到应用程序 …… (911)
44.2.3 以侦听器注册小程序 …… (912)
44.2.4 启动播放器 …… (913)
44.2.5 清除及停止播放器 …… (914)
44.2.6 综合 …… (914)
44.2.7 编译 BasicPlayer …… (917)
44.2.8 运行 BasicPlayer …… (917)
44.3 播放器的状态 …… (918)
44.4 向播放器添加控件 …… (919)
44.5 程序控制播放器 …… (922)
44.5.1 启动播放器 …… (922)
44.5.2 设置媒体时间 …… (922)
44.5.3 改变播放速度 …… (922)
44.5.4 改变音量 …… (923)
44.5.5 媒体播放器尺寸调整 …… (924)
44.5.6 添加进展条 …… (925)
44.6 连接多个播放器 …… (928)
44.7 创建自己的媒体流 …… (932)
44.7.1 被动媒体流 …… (932)
44.7.2 主动媒体流 …… (933)
44.8 一个大型应用程序 …… (933)
第 45 章 商业及 Java Wallet …… (943)
45.1 JCC 的安全性支持 …… (944)
45.2 商业消息 …… (944)
45.3 创建盒式带 …… (944)
45.3.1 CassetteControl 类 …… (945)
45.3.2 工具盒式带类 …… (947)
45.3.3 协议盒式带 …… (956)
45.3.4 操作盒式带 …… (963)
45.3.5 服务盒式带 …… (964)
45.3.6 用户接口盒式带 …… (967)
第 46 章 数据结构及 Java 工具 …… (979)
46.1 什么是数据结构 …… (980)
46.2 集成 …… (980)
46.2.1 集成接口 …… (980)
46.2.2 List 接口 …… (981)
46.2.3 Map 接口 …… (982)
46.2.4 Iterator 接口 …… (983)
46.2.5 ListIterator 接口 …… (983)
46.3 Vector 类 …… (983)
46.3.1 创建向量 …… (984)
46.3.2 向向量中添加对象 …… (984)
46.3.3 访问向量中的对象 …… (985)
46.3.4 Enumeration 接口 …… (986)
46.3.5 在向量中搜索对象 …… (986)
46.3.6 从向量中删除对象 …… (987)
46.3.7 改变向量的大小 …… (988)
46.4 HashTable 类 …… (988)
46.4.1 散列表中存储对象 …… (989)
46.4.2 从散列表中检索对象 …… (989)
46.4.3 从散列表中删除对象 …… (989)
46.5 Properties 类 …… (990)
46.5.1 设置属性 …… (990)
46.5.2 查询属性 …… (991)
46.5.3 保存和提取属性 …… (991)
46.6 Stack 类 …… (992)
46.7 Date 类 …… (993)
46.7.1 比较日期 …… (994)
46.7.2 转换日期为字符串 …… (994)
46.7.3 改变日期属性 …… (995)
46.8 BitSet 类 …… (995)
46.9 SringTokenizer 类 …… (996)
46.10 Random 类 …… (998)
46.11 Observable 类 …… (998)
第 47 章 java.lang …… (1008)
47.1 Java.lang 软件包 …… (1009)
47.2 Object 类 …… (1010)
47.2.1 检测对象相等 …… (1010)
47.2.2 对象的字符串表示 …… (1011)
47.2.3 克隆对象 …… (1011)
47.2.4 析构化 …… (1012)
47.2.5 系列化对象 …… (1013)
47.2.6 散列码 …… (1013)
47.2.7 wait( )和 notify( ) …… (1014)
47.2.8 获取对象的类 …… (1017)
47.3 Class 类 …… (1017)
47.3.1 动态加载 …… (1017)
47.3.2 获取关于类的信息 …… (1018)
47.4 Package 类 …… (1019)
47.5 String 类 …… (1020)
47.5.1 创建字符串 …… (1020)
47.5.2 字符串长度 …… (1022)
47.5.3 比较字符串 …… (1022)
47.5.4 搜索字符串 …… (1023)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

47.5.5 提取部分字符串 …… (1024)
47.5.6 修改字符串 …… (1024)
47.6 StringBuffer 类 …… (1025)
47.6.1 创建 StringBuffer …… (1025)
47.6.2 添加字符到一个 StringBuffer 中 …… (1025)
47.6.3 StringBuffer 的长度 …… (1026)
47.6.4 在 StringBuffer 中获取和设置字符 …… (1026)
47.6.5 从 StringBuffer 创建一个字符串 …… (1027)
47.7 Thread 类 …… (1027)
47.7.1 创建线程 …… (1027)
47.7.2 启动和结束线程 …… (1028)
47.7.3 等待线程完成 …… (1028)
47.7.4 休眠与让出 …… (1029)
47.7.5 守护线程 …… (1030)
47.7.6 线程优先级 …… (1030)
47.7.7 获得线程信息 …… (1031)
47.8 ThreadGroup 类 …… (1032)
47.9 Throwable 类 …… (1033)
47.10 System 类 …… (1034)
47.10.1 系统输入输出流 …… (1034)
47.10.2 获取当前时间 …… (1035)
47.10.3 退出虚拟机 …… (1035)
47.10.4 获得系统属性 …… (1035)
47.10.5 强制垃圾收集 …… (1036)
47.10.6 加载动态库 …… (1037)
47.11 Runtime 类和 Process 类 …… (1037)
47.11.1 查询可用内存 …… (1037)
47.11.2 运行外部程序 …… (1037)
47.12 Math 类 …… (1038)
47.12.1 min 和 max …… (1038)
47.12.2 绝对值 …… (1039)
47.12.3 随机数 …… (1039)
47.12.4 舍入 …… (1039)
47.12.5 幂和对数 …… (1040)
47.12.6 三角函数 …… (1040)
47.12.7 数学常量 …… (1041)
47.13 对象包装类 …… (1041)
47.14 Character 类 …… (1041)
47.15 Boolean 类 …… (1042)
47.16 Number 类 …… (1043)
47.17 Integer 类 …… (1043)
47.18 Long 类 …… (1044)
47.19 Byte 类 …… (1045)
47.20 Short 类 …… (1046)
47.21 Float 类 …… (1046)
47.22 Double 类 …… (1047)
47.23 Void 类 …… (1048)
47.24 java.math.BigInteger 类 …… (1048)
47.24.1 创建 BigInteger …… (1048)
47.25 java.math.BigDecimal 类 …… (1050)
47.25.1 创建一个 BigDecimal …… (1050)
47.26 ClassLoader 类 …… (1051)
47.27 SecurityManager 类 …… (1055)
47.28 Compiler 类 …… (1056)
第 48 章 反射 …… (1058)
48.1 什么是反射 …… (1059)
48.2 创建一个只知道构造器列表的类 …… (1060)
48.3 检查类的方法 …… (1065)
48.3.1 获得方法列表 …… (1065)
48.4 使用 getDeclaredMethod( )方法激活方法 …… (1068)
48.5 激活使用本地类型作为参数的方法 …… (1071)
48.6 获得一个类的声明字段 …… (1072)
第 49 章 使用其它语言扩展 Java …… (1074)
49.1 本地方法,Java 最终边界 …… (1075)
49.2 本地化 …… (1075)
49.3 JNI 精粹 …… (1076)
49.4 编写本地方法 …… (1076)
49.4.1 第一步:编写 Java 代码 …… (1076)
49.4.2 第二步:将 Java 代码编译成类文件 …… (1077)
49.4.3 第三步:产生 JNI 风格的头文件 …… (1077)
49.4.4 第四步:实现本地方法 …… (1078)
49.4.5 第五步:创建共享库 …… (1078)
49.4.6 第六步:运行 Java 程序 …… (1079)
49.5 从本地方法访问对象字段 …… (1079)
49.6 从本地方法访问 Java 方法 …… (1083)
49.7 访问静态字段 …… (1085)
49.8 访问静态方法 …… (1086)
49.9 本地方法的异常处理 …… (1086)
第 50 章 Java 与 C(++) …… (1088)
50.1 共同的祖先 …… (1089)
50.2 基本 Java 语法 …… (1089)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

50.2.1 词汇结构 …… (1089)
50.2.2 注释 …… (1089)
50.2.3 丢弃的内容 …… (1090)
50.2.4 运行时库 …… (1090)
50.3 Java 程序结构 …… (1091)
50.3.1 程序总体框架 …… (1091)
50.3.2 只有方法没有函数 …… (1091)
50.3.3 无预处理器 …… (1092)
50.3.4 源文件名 …… (1093)
50.4 Java 数据类型 …… (1093)
50.4.1 整型数据类型 …… (1093)
50.4.2 统一码字符 …… (1094)
50.4.3 布尔数据类型 …… (1094)
50.4.4 浮点类型 …… (1094)
50.4.5 集合数据类型 …… (1095)
50.4.6 类型转换与强制类型转换 …… (1095)
50.5 对象和类 …… (1095)
50.5.1 声明引用类型 …… (1095)
50.5.2 操作引用 …… (1096)
50.5.3 方法调用:通过值调用和通过引用调用 …… (1096)
50.5.4 原始数据类型及 java.lang 的包装类 …… (1097)
50.5.5 对象的生存周期 …… (1098)
50.5.6 Java 引用与 C++ 指针 …… (1099)
50.6 集合类型:字符串、数组及向量 …… (1099)
50.6.1 字符串 …… (1099)
50.6.2 数组 …… (1100)
50.6.3 向量 …… (1101)
50.7 类的层次与继承 …… (1102)
50.7.1 继承的语法 …… (1102)
50.7.2 instanceof 操作符 …… (1103)
50.7.3 继承和多态性 …… (1103)
50.7.4 接口与多重继承 …… (1103)
50.7.5 超级引用 …… (1104)
50.7.6 没有范围解析操作符 …… (1105)
50.8 语句 …… (1105)
50.8.1 循环 …… (1105)
50.8.2 条件语句 …… (1106)
50.8.3 同步语句 …… (1106)
50.8.4 操作符与表达式 …… (1106)
50.9 名字空间 …… (1107)
第八部分 调试 Java …… (1109)
第 51 章 调试 Java 代码 …… (1111)
51.1 sun.tools.debug 包的体系结构 …… (1112)
51.1.1 客户/服务器程序调试器管理 …… (1112)
51.1.2 特殊类型 …… (1117)
51.1.3 本地类型 …… (1124)
51.1.4 堆栈管理 …… (1125)
51.1.5 线程管理 …… (1127)
51.1.6 总结 …… (1130)
51.2 深入探讨 JDB …… (1131)
51.2.1 基本结构 …… (1131)
51.2.2 JDB 命令行 …… (1132)
51.2.3 JDB 输入文件 …… (1134)
51.2.4 JDB 命令集 …… (1134)
51.2.5 一般命令 …… (1136)
51.2.6 上下关系命令 …… (1138)
51.2.7 信息命令 …… (1139)
51.2.8 断点命令 …… (1143)
51.2.9 异常命令 …… (1145)
51.2.10 线程命令 …… (1145)
51.2.11 JDB 总结 …… (1146)
第 52 章 理解 .class 文件 …… (1148)
52.1 基本度量 …… (1149)
52.2 .class 文件的元素 …… (1149)
52.3 定义 …… (1149)
52.3.1 常量池 …… (1150)
52.3.2 类型信息 …… (1153)
52.3.3 属性 …… (1154)
52.4 .class 文件的结构 …… (1155)
52.4.1 类标志字段 …… (1156)
52.4.2 字段信息结构 …… (1156)
52.4.3 ConstantValue 属性 …… (1157)
52.4.4 方法信息结构 …… (1158)
52.4.5 SourceFile 属性 …… (1161)
52.5 现在可以做些什么 …… (1161)
第 53 章 Java 虚拟机的内部机制 …… (1163)
53.1 JVM 的元素 …… (1164)
53.1.1 虚拟机的体系结构 …… (1164)
53.1.2 内存管理和垃圾收集 …… (1166)
53.1.3 类文件验证 …… (1169)
53.1.4 JVM 字节码 …… (1170)
第九部分 JavaScript …… (1181)
第 54 章 Java 与 JavaScript …… (1183)
54.1 Java 与 JavaScript …… (1184)
54.2 JavaScript 并非 Java …… (1184)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

54.3 解释与编译 …… (1185)
54.4 基于对象和面向对象 …… (1188)
54.5 强类型和松散类型 …… (1188)
54.6 动态绑定和静态绑定 …… (1189)
54.7 磁盘访问限制 …… (1189)
54.8 不同功能(范围限制)及与 HTML 的代码集成 …… (1190)
54.9 快速发展与相对稳定 …… (1192)
54.10 库 …… (1193)
54.11 JavaScript 和 Java 集成 …… (1194)
第 55 章 JavaScript 入门 …… (1195)
55.1 基础 …… (1196)
55.2 第一个脚本 …… (1197)
55.3 事件 …… (1197)
55.4 使用事件句柄 …… (1198)
55.5 变量 …… (1199)
55.6 变量名 …… (1200)
55.7 变量的作用范围 …… (1201)
55.8 常量 …… (1201)
55.9 表达式和操作符 …… (1201)
55.10 控制语句 …… (1203)
55.10.1 条件语句 …… (1203)
55.10.2 循环语句 …… (1205)
55.10.3 break 和 continue 语句 …… (1206)
55.10.4 注释语句 …… (1207)
55.11 JavaScript 的函数 …… (1208)
55.12 数组 …… (1210)
55.13 内置函数 …… (1211)
55.14 对象 …… (1211)
55.14.1 圆点指示符 …… (1212)
55.14.2 方法与属性 …… (1212)
55.14.3 Window 对象 …… (1212)
55.14.4 Document 对象 …… (1213)
55.14.5 Form 对象 …… (1214)
55.14.6 Navigator 对象 …… (1215)
55.14.7 String 对象 …… (1215)
55.14.8 Math 对象 …… (1216)
55.14.9 Date 对象 …… (1217)
55.15 一个实例 …… (1218)
第十部分 Java 资源 …… (1221)
第 56 章 Java 资源 …… (1223)
56.1 Web 站点 …… (1224)
56.1.1 Earthweb 的 developer.com …… (1224)
56.1.2 Focus on Java(Java 焦点) …… (1224)
56.1.3 Inside Java …… (1224)
56.1.4 Java Applet Rating Service (JARS) …… (1224)
56.1.5 Java Boutique …… (1225)
56.1.6 Java Developer's Journal (Java 开发人员杂志) …… (1225)
56.1.7 Java Developer Connection (Java 开发者连接) …… (1225)
56.1.8 Java Lobby …… (1225)
56.1.9 Java Resources from Netscape (来自 Netscape 的 Java 资源) …… (1225)
56.1.10 Java World …… (1225)
56.1.11 JavaBeans 站点 …… (1226)
56.1.12 Sun 公司的主页 …… (1226)
56.1.13 Javology——Java 新闻及观点在线电子杂志 …… (1226)
56.1.14 微软的 Java 主页 …… (1226)
56.1.15 Swing Connection …… (1226)
56.1.16 TeamJava …… (1226)
56.2 新闻组 …… (1227)
56.3 邮件清单 …… (1227)
56.4 对移植问题的支持 …… (1228)
56.4.1 Amiga 移植问题 …… (1228)
56.4.2 DEC Alpha OSF/1 移植 …… (1228)
56.4.3 Linux 移植问题 …… (1228)
56.4.4 NEXTSTEP 移植问题 …… (1229)
第十一部分 附录 …… (1231)
附录 光盘的内容 …… (1233)
1 本书的实例代码 …… (1234)
2 第三方软件 …… (1234)
3 赠送软件 …… (1234)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第一部分　Java 概述

第 1 章　Java 能做什么
第 2 章　Java 设计
第 3 章　安装并启动 JDK
第 4 章　JDK 工具

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 1 章　Java 能做什么

本章内容：

1.1　Java 应用程序的几种类型
1.2　了解 Java 语言
1.3　Java 开发工具
1.4　Java 小应用程序
1.5　Java 图形用户界面应用程序
1.6　Java 命令行应用程序
1.7　Java 是客户/服务器方式的
1.8　如何跟上形势

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

1.1 Java 应用程序的几种类型

到现在为止你可能已经了解了大量关于 Java 的赞誉，所以走到街上买回了这本书。其实这些赞誉是恰如其分的，Java 的方方面面确实是像你听到的那样。

在本章里通过浏览几个可以自己使用 Java 开发的小应用程序来审查一下 Java 语言提供的各种可能性，为了理解方便可以浏览几个目前互联网上正在运行的几个 Java 小应用程序示例和 Java 命令行应用程序示例。到看完本章内容后你应该拥有一些使用 Java 干些什么的好主意，并会因为 JDK 1.2 为这一语言所作的令人难以置信的完善和 Java 正在改变计算世界而感到振奋。

Java 不仅仅是用于创建小应用程序和服务器端应用程序的编程语言。令人惊讶的是它能够被用来创建种种巨大的应用程序。下面是用户可以使用 Java 语言建造的几种 Java 应用程序：

Applets(小应用程序)

GUI(图形用户界面)应用程序

命令行应用程序

Servlets(服务器端应用程序)

包(库)

嵌入式应用程序(像示波器和其它嵌入式计算机)

笔基程序

小应用程序是基本的应用程序，它运行在支持 Java 的浏览器上，比如 Netscape 的 Navigator，Microsoft 的 Internet Explorer 或 Hot Java。

使用 Java 开发的 GUI 应用程序拥有图形界面并且是独立自主的。他们能够像其它的 GUI 应用程序一样单独运行，比如 Windows 的 Notepad 应用程序，它不需要调用网络浏览器来运行。

命令行应用程序可以在 MS-DOS 命令提示符下或 UNIX 外壳提示符下运行，就像 MS-DOS 的 xcopy 命令和 UNIX 上的 ls 命令一样。

包本身并不是应用程序，它就像是一个属于一个系统(类似于 C++ 的类库)的类(可移植的 Java 字节码文件)的集合。包没有像用在各种操作系统上的动态库那种定制的格式，Java 实现更加简单和具有更强大的可移植性。

基本上属于一个包的类都被放在同一个目录下，例如所有属于 Java 抽象窗口工具(AWT)包 Java.awt 的类都放在 C:\jdk 1.2\src 下名为 AWT 的目录下。下面是以由 Java 开发工具提供的各种包构成的目录树。

```
c  java classes
    applet
    awt
        Button class
        Color class
        Event class
    io
    lang
    net
    util
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

作为示意，上面的图显示了 AWT 目录下的几个类文件(其实在 AWT 目录下有几打类文件)。

1.2　了解 Java 语言

早在 Java 诞生之初，Sun Microsystems 公司就发表了一份“白皮书”，其中使用了一连串的时髦话，听起来令人头晕目眩。

Java 是简单的、面向对象的、稳定的、安全的、高效的、与平台无关的、解释型的、多线程的、动态的语言。

唷！试着一口气说出这些看感觉如何？无论感觉怎样，在接下来的本书内容中将详细地研讨该语言，但在本章中人们需要了解的一个词是：“解释型的”。

Java 源代码编译成可移植的字节码后需要有解释器来运行它们。对于小应用程序来说这个任务是由浏览器来完成的。对于 GUI 和命令行应用程序则需要 Java 解释器程序来执行它们。“Java 命令行应用程序”中的示例阐述了这两种方法。

1.3　Java 开发工具

Java 流行的原因不仅是因为语言本身的优点。同样也由于绑定在 Sun Microsystems 提供的 Java 开发工具(JDK)上的丰富的包(对于 C++ 程序员来说类似于类库)使得它如此流行。这些预先编写的对象使得用户可以快速地编写和运行 Java 程序，这其中有两个主要的原因：

- 用户不需要开发他们已经提供了的功能。
- 这些原代码运用于所有应用程序。

下面是对 Java 提供的几个较重要的包的简要描述：

包	描述
java.applet	开发小应用程序的类
java.awt	开发诸如窗口对话框按钮，文本域等图形用户界面的抽象窗口工具类。
java.net	用于网络、统一资源定位器(URL)、客户/服务器套接字的类。
java.io	用于各种输入输出的类。
java.lang	用于各种数据类型、运行进程、字符串、线程等方面的类。
java.util	用于日期、向量等方面的实用类。
java.awt.image	管理和操作图像的类。

1.4　Java 小应用程序

就像前面提到的一样，Java 小应用程序是运行在支持 Java 的网页浏览器中的。因为原来的网页浏览器是为了显示 HTML 文档而开发的，网络浏览器需要使用一个 HTML 标志来调用小应用程序。这个 HTML 标志是〈APPLET〉，如下面的例子所示：

〈applet code = TextEdit.class　width = 575　height = 350 > < /applet〉

在第 14 章和第 15 章将详细地研究小应用程序。

网络上真实的 Java 小应用程序例子

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

因为图片胜过千言万语,所以用户可能喜欢看一下当今网络上真实的 Java 小应用程序的示例。图 1.2 到图 1.11 都是小应用程序的例子。

图 1.1 显示了一个名为 NetProphet 的 Java 应用程序。NetProphet 是一个奇妙的实用工具,它可以用表格和图像来显示所有的股票。它是一个展示如何通过 Java 客户与服务器交互来产生动态信息的奇妙示例。NetProphet 可以从 http://www.neural.com/Netprophet/Netprophet.html 的 Neural Application 处获得。

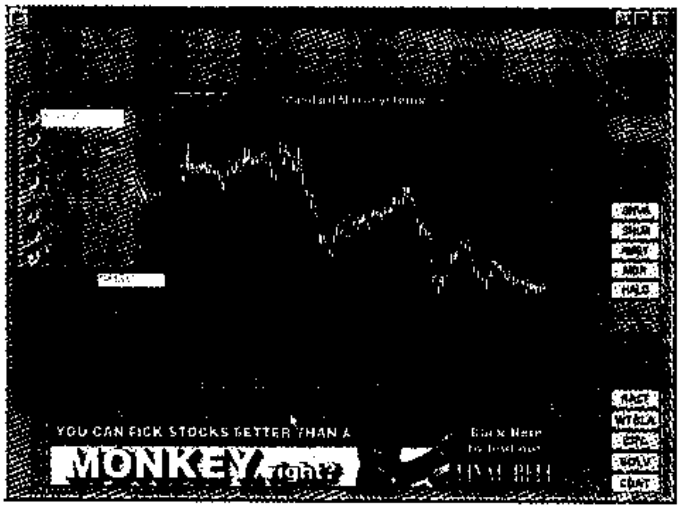


图 1.1 NetProphet 是一个用 Java 开发的客户/服务器应用程序的示例

图 1.2 显示了 Blossom Flowers 公司(http://www.blossomflowers.com/shopping_frame.html)使用的东部数据系统公司开发的网上购物的小应用程序(http://www.eastland.com/shoptech.html)。

这是一个非常好的小应用程序,因为它在互联网上使用了拖放特性。

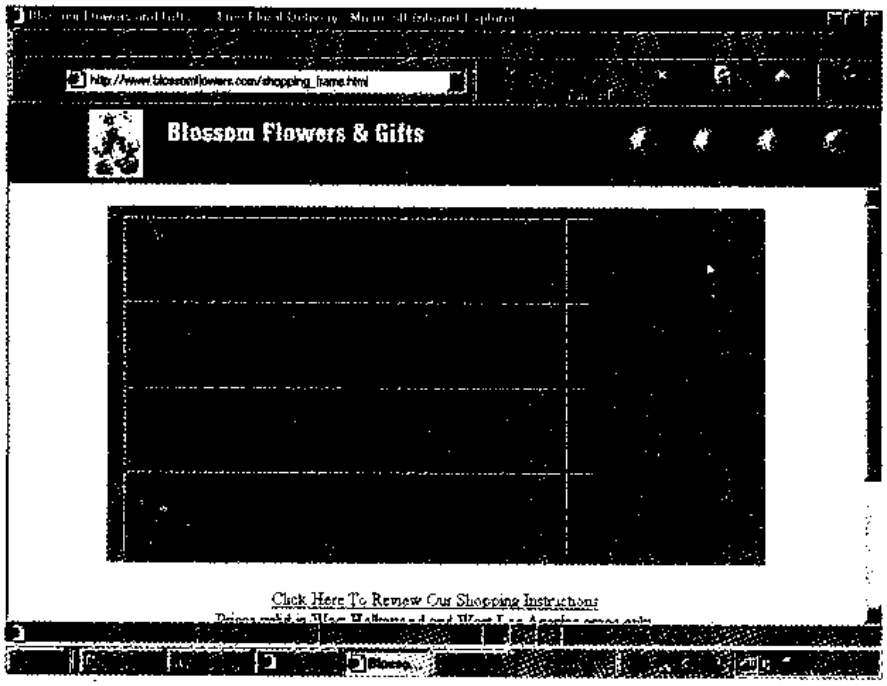


图 1.2 Eastland 公司开发的网上购物小应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在编写互联网游戏方面也能显示出 Java 是一个伟大的语言。图 1.3 显示的是我们娱乐了达几年之久的著名的俄罗斯方块游戏。这是一个使用 Java 开发的功能完整的俄罗斯方块。人们可在互联网的 htpp://www.tdb.uu.se/~karl/java/rubik.html 站点上玩这个游戏。

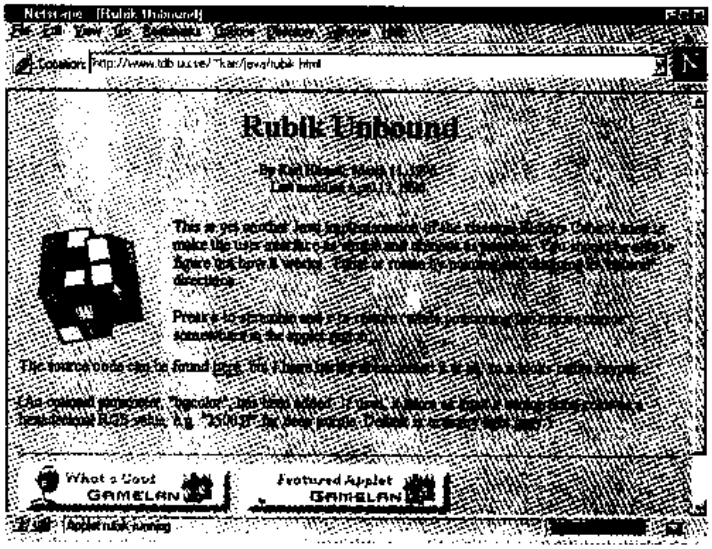


图 1.3　俄罗斯方块

图 1.4 显示了另一个使用三维图形的游戏。Palle Pedalpost(http://www.zapper.kd/postpil/postgame.html)使用了平滑的动画,很值得一看。



图 1.4　Pelle Pedalpost

莲花公司在使用 Java 语言开发 eSuite 程序时很有创意。eSuite 是一个构造组件应用程序的工具集成,如图 1.5 所示。

JDK 1.2 的奇妙扩展之一是使应用程序具有多媒体显示能力。图 1.6 显示的是一个用 Java 程序播放的 MPEG 电影。

芝加哥论坛使用 Java 的服务器端应用程序创建了一个奇妙的站点,名叫 Metromix。如

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

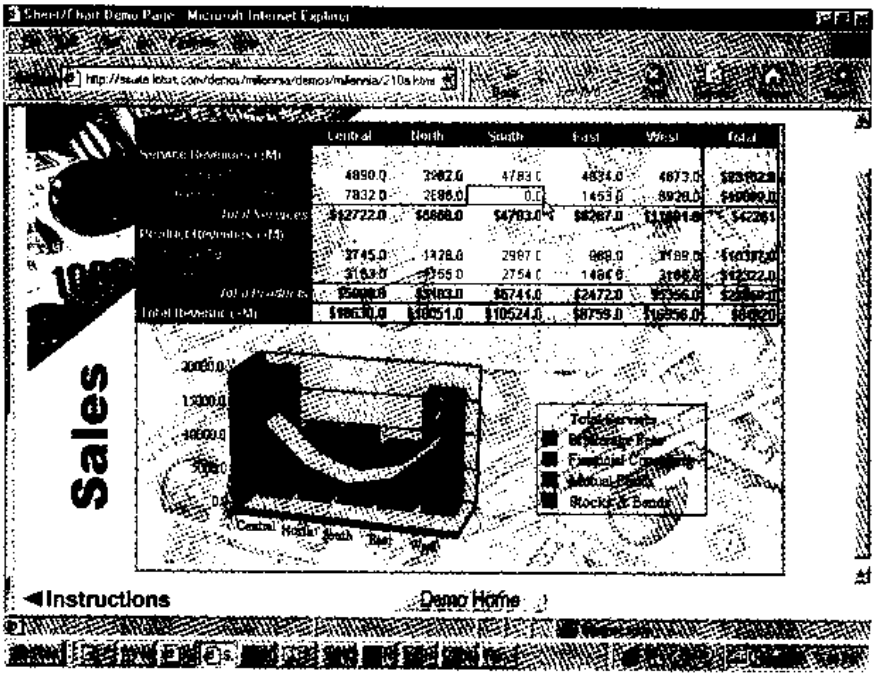


图 1.5 莲花公司的 eSuite



图 1.6 这个 MPEG 电影是用 Java 程序播放的

图 1.7 所示，Metromix 是芝加哥地区文娱节目预告。

来自哈佛大学的一帮学生创建了一个很有新意的课程安排系统。如图 1.8 所示，这个系统可以在 http://www.digitas.harvard.edu 站点上看到。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

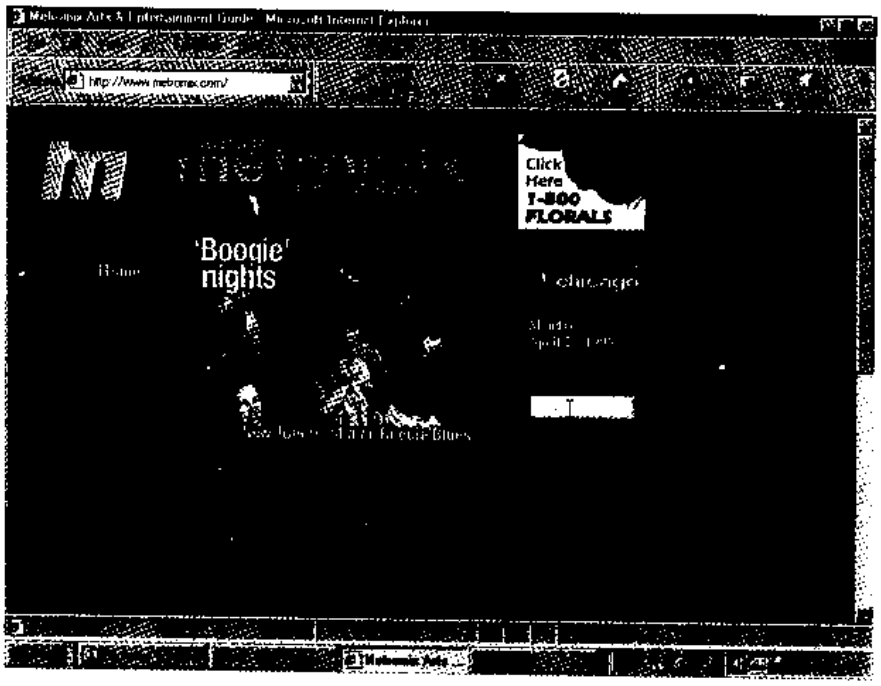


图 1.7　Metromix 网络站点

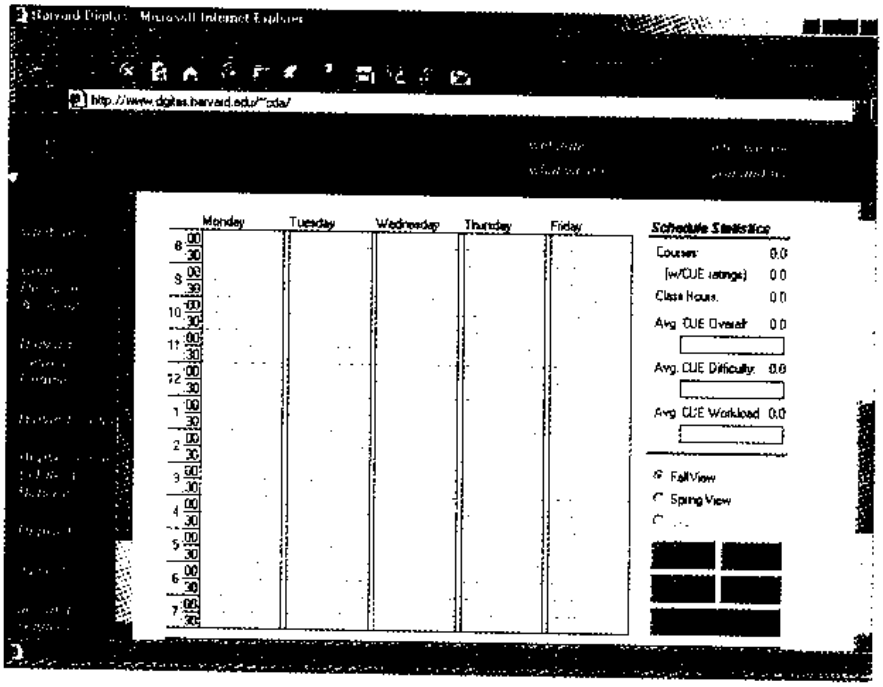


图 1.8　Digitas 课程选择助手

1.5　Java 图形用户界面应用程序

尽管 Java 小应用程序拥有很多闪光点，但是 Java 远远不只是小应用程序。Java 可以跨越所有支持平台开发可移植的图形用户界面应用程序。

为了阐明这一点请看一个名为 Text Editor(文本编辑器)的应用程序，这个程序是专门为简述上述观点而开发的。就像该程序名字所暗示的那样，该程序是用于编辑文本文件的，很像 Windows 的 Notepad(记事本)应用程序。图 1.9 显示的是 Text Editor 的小应用程序版本，图

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

1.10 显示的是 Windows 95 操作系统上的应用程序版本，而图 1.11 显示的是 Solaris 操作系统上的应用程序版本。

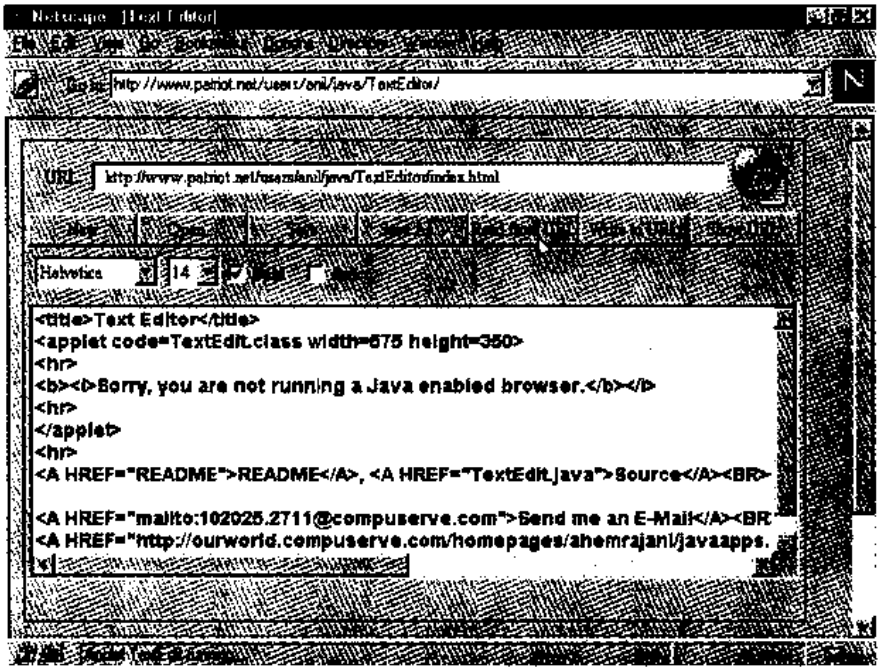


图 1.9 作为小应用程序运行的 Text Editor(文本编辑器)

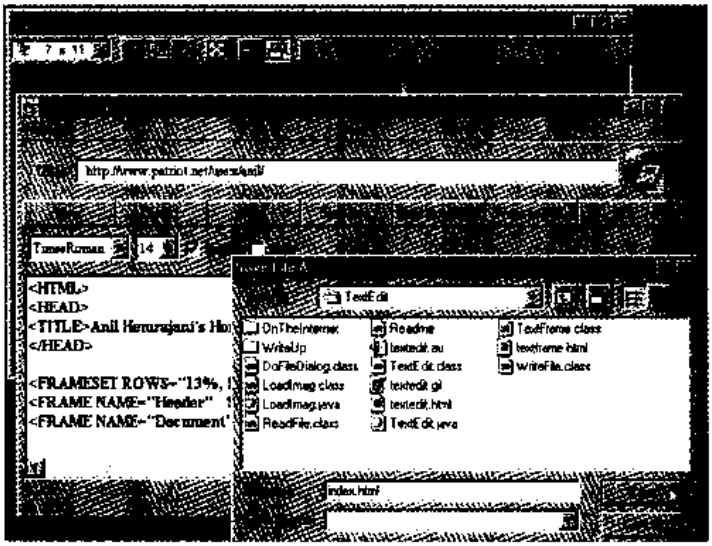


图 1.10 在 Windows 95 上使用 Java 解释器运行的
Text Editor(文本编辑器)应用程序

上述三种版本的 Text Editor 都是使用同一个 Java 源文件生成的。事实上这三个版本都是使用同一个字节码文件执行的，该字节码文件是在 Windows 95 下一次编译完后，拷贝到 Solaris 操作系统上的；在 Solaris 操作系统上无需再编译即可执行。

请注意 Java 解释器是如何运用于在 MS-DOS 提示符下的执行应用程序的。请注意图 1.10 和 1.11 中的文件对话框。如果读者是 Windows 95 或 Solaris 的用户，应该知道它们是用在这些操作系统上的标准文件对话框。作为开发者，不需要定制任何代码便可在本地得到所见即所得的效果。所有需要做的事情就是保证小应用程序或应用程序要调用的类(字节码)文件可用，剩下的事情(本地所见即所得的效果和系统的特殊特性等)是由 Java 动态链接库完成的。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

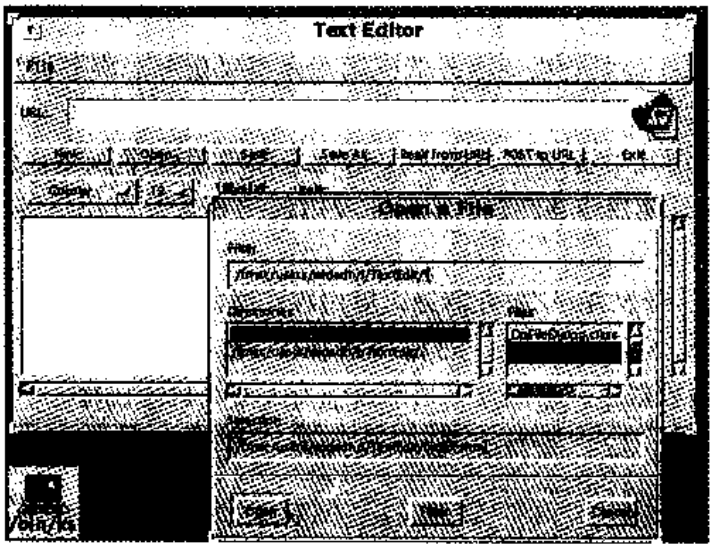


图 1.11　在 Solaris 上运行的 Text Editor(文本编辑器)应用程序

1.6　Java 命令行应用程序

尽管当今世界上 GUI 应用程序已成为各种计算机平台上的标准,但仍有人有时需要使用命令行来执行某些任务。这时就可以使用 Java 提供的开发命令行应用程序的功能。

命令行应用程序与图形用户界面应用程序之间唯一的不同在于命令行应用程序不使用任何由 Java 提供的图形用户界面的特性,换句话说:命令行应用程序不使用 java.awt 包。

图 1.12 显示的是一命令行应用程序的示例,该程序名为 copyURL,从本质上说是一个拷贝实用程序,它可以从互联网上拷贝东西到本地磁盘上。这个应用程序使用了 java.net 包以便获取互联网上关于资源(文件)的信息。copyURL 还使用了 java.io 包以便读取互联网上文件中的数据字节并将其写入本地文件中。

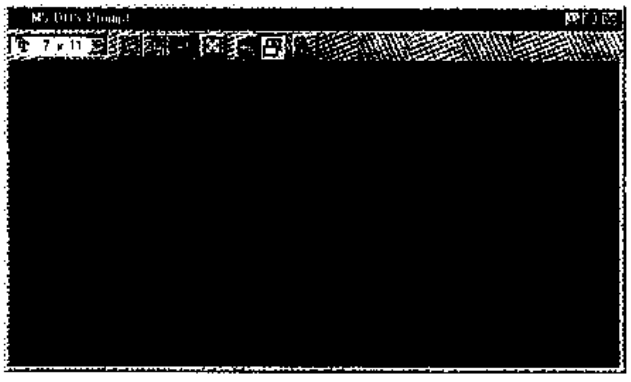


图 1.12　copyURL.class 是一个互联网命令行拷贝实用程序

1.7　Java 是客户/服务器方式的

在当今的计算机世界里,客户/服务器技术在大多数公司里都占有一席之地。该技术最大

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

优点是计算机处理的负担是分摊在客户和服务器上的。客户可能是任意的应用程序(图形用户界面应用程序、Telnet 等),它需要服务器应用程序提供服务。服务器程序可以是数据库服务器、应用程序服务器和通信服务器等等。

在本章里将要介绍几个 Java 客户端小应用程序和应用程序。Java 也提供服务器端处理类。Java 小应用程序只能用作客户端,而 Java 应用程序则是既可用于客户端,也可以用于服务器端。

Java.net 包提供了开发客户/服务器方式应用程序所需要的类。图 1.13 显示的 Java 小应用程序 javaSQL,它发送由用户输入的自由表 SQL 查询给服务器应用程序 javaSQLd 。javaSQLd 交替地查询数据库并返回查询结果给 javaSQL 小应用程序 。

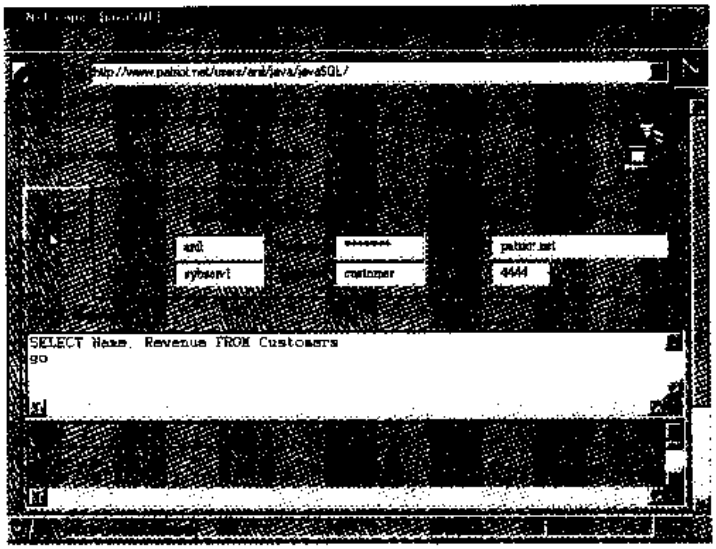


图 1.13 JavaSQL 客户小应用程序

图 1.14 简述了 javaSQL 和 javaSQLd 之间的关系。想象一下坐在家中通过支持 Java 的浏览器直接查询远程数据库是怎样的情形。使用 Java 可以有无边无际的可能性。



图 1.14 JavaSQL 和 JavaSQLd

1.8 如何跟上形势

与网络有关的技术正在日新月异地发展,很难说谁正站在巅峰,互联网更新的步伐更是极端迅速,而 Java 也正以这种令人震惊的速度发展着。

跟上形势的唯一办法是访问发布 Java 新闻和 Java 示例的互联网站点。互联网上有数十个站点提供应时的信息,读者可以在第 56 章"Java 资源"看到由提供 Java 信息有一定历史的站点组成的列表。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 2 章　Java 设计

本章内容：

2.1　解释 Java

2.2　Java 是面向对象的

2.3　Java 虚拟机

2.4　安全性和 Java 虚拟机

2.5　Java API

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

2.1 解释 Java

在使用 Java 编写程序和小应用程序之前弄清楚 Java 是如何工作的很重要。本章将介绍语言的限制(有意的和无意的)和为什么代码可重用。

虽然 Java 既要解释又要编译,但严格地讲 Java 是由浏览器解释的。事实上只有大约百分之二十的 Java 代码由浏览器解释,但这是关键的百分之二十。Java 的稳定性和能够在多平台上运行的能力都是在最后的本地编译阶段赋予的。

程序员首先使用 Java 编辑器把 Java 源代码编译成字节码。这些字节码是二进制的,并且是与平台无关的(在任何平台上都可以很好地工作)。仅有字节码还不够,只有将其与 Java 运行环境(一般指的是浏览器)结合才能正常工作。因为每个 Java 运行环境都针对特定的平台,所以 Java 程序必须通过运行环境的解释才能在特定平台上运行。

对于开发人员来说 Java 的平台特殊特性是一个好消息。它意味着 Java 就是纯粹的 Java 代码,不用关心是为哪个特定的平台开发的应用程序。用户可以在 UNIX 系统下编写和编译 Java 小应用程序并将这个小应用程序嵌套在网页中。不同的三个人在不同的机器、不同的环境下运行,运行同一个小应用程序。只要三个人都运行支持 Java 的浏览器就可以,不管是在 IBM 机器上还是在 HP 机器上或者是在 Macintosh 机器上其运行结果都是一样的。使用 Java 就意味着只有 Java 源代码需要维护,只要将其编译成字节码就可以在各种平台上运行。无须使用针对运行平台的编译器,对于程序员来说无疑是一个好消息。

解释带来的缺点就是执行效率的降低,这是由于在运行之前需要浏览器处理一些类文件(翻译它们)所致。在传统的程序里(比如 C++)生成的代码可以直接在计算机上运行。解释引起的效率降低意味着 Java 程序只能以 C++ 程序 1/2 到 1/6 的速度运行。

使用一个名为即时(just-in-time,简称 JIT)的编译器可以大大弥补低效的缺陷。即时编译器是将 Java 方法编译成用于相应平台的本地代码。它被嵌套在针对平台的 Java 环境(比如 Netscape)中,未经 JIT 编译器编译的方法不能被翻译成本地码,仍然是平台无关的字节码。这些字节码由 Java 虚拟机解释到任意一种平台上。Java 应用程序是可移植的,但是即时编译器是不可移植的,因为它产生的本地码是针对某种平台的。严格地讲,不同的平台需要不同版本的虚拟机。一般情况下用户不需关心 JIT。Netscape 公司的 Navigator 浏览器和 Microsoft 公司的 Internet Explorer 浏览器都包含有 JIT 编译器。

为什么说编译和解释的结合是一种积极的特性呢?

- 它促进了安全性和稳定性。Java 环境包含一个叫做链接器的组成部分,它既检查进入计算机的数据,保证数据中不含有有意危害文件的内容(安全性),又检查进入计算机的文件是否会破坏计算机的系统功能(稳定性)。
- 更重要的是解释与编译的结合可以消除用户对版本不匹配的担心。

事实上,编译工作的最后部分是由与平台无关的设备完成的,这种设备是由最终用户维护的,这种做法可以减少用户对多平台的资源的维护。解释还可以使得数据在运行时刻得到合并,这一点是动态行为的基础。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

2.2　Java 是面向对象的

Java 是面向对象的语言。因此它把数据定义为对象，并使用方法来操作对象。正如我们解释的那样，Java 和 C++ 有许多共同的底层原则，它们的差异在于风格和结构。简单地讲面向对象的编程语言（简称 OOP）描述的是对象之间的相互作用。

参见第 50 章“Java 与 C++”可以更多地了解 Java 和 C++ 的相似处。

许多面向对象的语言支持多继承性，它有时可能导致混乱和不必要的编译。Java 不是这样。正如哲学中的“更少就是更高”的思想一样，Java 只支持单继承性，这就意味着每个类同时只能继承一个其它的类。这种类型的继承避免了类继承多个行为矛盾的或行为互斥的类的问题。Java 可以创建完全抽象的类，比如接口。接口允许定义可能与其它类共享的方法，而不用管其它类如何处理该方法。

参见第 5 章“面向对象编程”。

> **注意：**虽然 Java 不支持多继承性，但是 Java 却允许某个类用在多个接口上。

每个类，不管抽象的还是不抽象的，都是使用方法集来定义行为的。Java 的所有代码都可以分成类。方法可以从一类继承到另一个类，在类的层次结构的顶层是一个叫做 Object（对象）的类，这个类属于 Java 核心 API 的 Java.lang 包。在本章的最后部分将介绍 Java 的核心 API。

参见第 11 章“类”可以更多地了解类和对象。

对象可以实现任意多个接口（或抽象类）。Java 的接口很像接口定义语言（IDL）里的接口。这种类似意味着很容易建立由 IDL 到 Java 的编译器。这个编译器可以用在对象的公共对象请求代理结构（CORBA）系统中以建立分布对象系统。这样好吗？IDL 接口和 CORBA 系统都可以用在各种变异的计算机系统中，这种变易有助于 Java 的平台无关性。

参见第 42 章“JavaIDL：Java 的 CORBA 的接口”。

为了努力保持 Java 的简单，Java 的面向对象语言并非一切都是对象。布尔型、数字型和其它简单的类型都不是对象，但是 Java 为所有简单类型都提供了封装对象。封装对象使所有简单类型用起来像类一样。

记住 Java 是面向对象的是很重要的，它不允许用户声明任何在对象中没有封装的东西。尽管 C++ 被认为是面向对象的语言，但是它允许存在一些不良的开发习惯和不封装类型。

参见第 7 章“数据类型及其它标记”，以便更多地了解类型。

面向对象设计允许即插即用。Java 面向对象的工具在 C++ 工具的基础上扩充了许多动态的方法。

2.3　Java 虚拟机

Java 的心脏是 Java 虚拟机，即 JVM。Java 虚拟机是只能驻留在内存中的虚拟计算机。JVM 使 Java 程序在各种平台上运行就像是在编译代码的平台上运行一样。正因为 Java 程序是为 Java 虚拟机编译的，所以它成为不平凡的语言，但是为了使 Java 程序能够运行在某个特

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

定的平台上,该平台必须启用 Java 虚拟机。

参见第 53 章"Java 虚拟机的内部机制"可更多地了解 JVM。

正是因为 Java 虚拟机,才使得 Java 是可移植的,它提供了一个介于编译过的 Java 程序和底层硬件平台及操作系统之间的一个抽象层。

运行 Java 虚拟机只占用很少内存。这样设计的目的是:可以将其用在各种电器上。事实上,Java 语言最早是为家用电器开发的。电话、PC 机、电视机等等在不久的将来都会固化有允许 Java 程序运行的 Java 虚拟机,酷吧?

2.3.1 Java 源代码

Java 源代码被编译成的代码是字节码级的,而不是位码级的。Java 虚拟机执行的是 Java 字节码。Javac 程序是 Java 的编译器,它读取带有 .java 扩展名的文件,将文件中的源代码转换成字节码并使用带有 .class 扩展名的文件存储生成的字节码。

Java 虚拟机从 .class 文件中读取字节码流作为指令序列,每个指令用一个字节操作码组成,它是一个特殊的可识别的命令、零或多个操作符(需要数据的执行操作码)。操作码告诉 Java 虚拟机干什么。如果 Java 虚拟机不仅只需要操作码来执行某个行为,那么操作符就紧跟在操作码后面。

参见第 52 章"理解 .class 文件"可以更多地了解操作码。

JVM 包括四个部分:

- 堆栈
- 寄存器
- 无用单位收集堆
- 方法区

2.3.2 Java 堆栈

Java 虚拟机的地址是 32 位的,因此它能够寻址 4G 内存空间,每个内存地址存放一个字节的信息。每个 JVM 寄存器存放一个 32 位的地址。堆栈、无用单元收集堆和方法区都驻留在这 4G 的内存空间里。这 4G 的内存寻址限制现在不是真正的限制,因为许多 PC 机上的内存(RAM)不超过 32M。Java 里的每个单个方法都限制在 32M 以内。

2.3.3 Java 寄存器

所有进程都使用寄存器。Java 虚拟机使用下列寄存器管理系统堆栈:

- 程序计数寄存器　跟踪程序执行的准确位置。
- 堆栈指针寄存器　指示操作栈顶。
- 框架寄存器　指向当前执行的环境。
- 变量寄存器　指向当前执行环境中第一个本地变量。

Java 开发组决定 Java 只使用四个寄存器,这是因为如果使用的寄存器数多于处理器端口数,那么处理器的效率将严重地降低。

Java 虚拟机中的堆栈用于存放变量,Java 程序向 Java 虚拟机传递字节码,Java 虚拟机为每个方法创建堆栈框架,每个框架维护三种信息:

- 局部变量　由变量寄存器指向的 32 位变量数组。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

- 执行环境　由框架寄存器指向和执行的方法。
- 操作堆栈　执行先进先出规则(FIFO)。它是 32 位宽度的,它为操作码维护必要的参数。该堆栈的顶部是由堆栈指针寄存器索引的。

2.3.4　无用单元收集堆

堆是系统为类的实例分配的内存集合。无论何时使用 new 操作符分配内存都是从该堆中获取内存。可以直接调用无用内存收集器,在许多情况下这并不是必须的和提倡的做法。运行环境跟踪对堆栈中收集每个对象的引用,并且自动释放不再引用的对象所占用的内存。无用内存收集是作为线程在后台运行的,当 CPU 不忙时,CPU 执行清理任务。

2.3.5　Java 的方法区

Java 虚拟机还有另外两个内存区域:

- 方法
- 常量池

至于这两个区域实际存在于哪里并没有限制,这样做是为了保证 Java 虚拟机的可移植性和安全性。实际上这个内存区域可以存在于任何地方,这样黑客就不能伪造指针,从而保证了安全性。

Java 处理下列几种数据类型:

- 字节(8 位)
- 浮点(32 位)
- 整数(32 位)
- 短整数(16 位)
- 双精度数(64 位)
- 字符(16 位)
- 长整数(64 位)

2.4　安全性和 Java 虚拟机

这一部分由六个子部分组成,在这部分里将一般性地探究网络计算机的安全问题并定义与可执行内容有关的安全性问题。通过六个步骤建立灵活可变的安全机制。因为技术上的缘故,有些公开的与安全性有关的问题仍在网上和其它媒体上被争论着。

2.4.1　可执行内容与安全性

在这一子部分中我们将分析网络上在一般交互联系过程中的安全性的概念和通过可执行的内容来实现安全性。

考查一下网络上的安全与交互的对偶性,然后再考察作为具有对偶性联系的介质的网络演变。为了做到这一点,让我们在可执行内容范围内给安全性下一个定义。

2.4.1.1　用户安全性问题的定义

其它机器上的程序若想通过网络进入用户机器,必须取得用户的信任和许可,方能得到存

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

取用户计算机上资源的权限,以及达到他的目的。由于程序是由他人编写的,所以不受用户限制。如果某人是黑客,那么他编写的可执行内容可能含有恶意的程序,一旦骗得信任即可像本地程序一样对本地资源进行操作。

参见第 34 章"Java 安全性深析"可以更多地了解 Java 的安全性。

那么用户就因此而完全限制其它外部程序访问本地任何资源吗?当然不能。这样做将从根本上削弱可执行内容的能力。较完整的可行的安全性解决策略包含以下六个步骤:

1.预料所有潜在的有害行为和攻击方案。

2.把所有这种有害行为减到最低程度。

3.构造一个隐含禁止有害行为的编程环境/计算机语言,以此来禁止潜在的有害行为。

4.逻辑地、公理地(如果可能的话)证明语言/环境对于预谋攻击方案是真正安全的。

5.用被证实是安全的语言实现可执行内容。

6.设计一个可被翻新为基本安全机制的语言,在未来一旦新的攻击方案产生时能够缺省地以一系列对抗措施来响应。根据前面的解决策略,与可执行内容有关的安全性问题可以分成下面六个子问题:

- 什么是潜在目标源和相应的攻击方案?
- 什么是基本的、最小的与攻击方案有关的行为组件集?
- 隐含禁止有害行为基本集的计算机语言/编程环境是如何设计的?
- 如何证明这样的语言/环境的安全性是像被声明的那样?
- 如何保证进入本地机器的可执行内容确实是用这种可信的语言编写的?
- 如何能够使得语言将来保证(可扩展性)增加安全策略以对付将来出现的威胁?

在后面部分可以了解到,Java 被设计成能够大部分(可能不是全部)地解决上述提到的安全性问题。在我们继续介绍 Java 安全性体系结构之前,我们先来介绍攻击目标和方案的识别。

2.4.1.2 潜在的缺陷

在这个子部分里,我们列出了各种可能的攻击方案和在用户的计算机上可能被有潜在威胁的、外部的可执行模块列为攻击目标的资源。

攻击方案可能是下述所列种类之一,并攻击其中所述目标之一(这并不是个详尽无遗的列表):

- 破坏或修改数据的完整性和/或程序的执行状态。
- 收集和窃取秘密数据。
- 锁定资源,使得合法用户和程序无法使用。
- 未经授权窃取资源后在外部使用。
- 引起非致命的效率降低,尤其是输出设备。
- 盗用用户的识别码(IP),假冒用户的计算机身份去攻击网络上的其它目标。

表 2.1 列出了可能成为潜在目标的资源和可能遭受攻击的类型。好的安全策略为每个资源设定安全/风险权并设计一套适当的对策来应付外部可执行的内容。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

表 2.1　潜在目标和攻击

目标	破坏完整性	窃取信息	锁定拒绝使用	窃取资源	非致命破坏	假冒用户
文件系统	×	×	×	×	×	
秘密数据	×	×	×		×	×
网络	×	×	×	×		×
CPU		×	×	×		
内存		×	×	×		
输出设备	×	×	×	×	×	
输入设备		×	×		×	
操作系统,程序状态	×		×	×	×	

2.4.2　Java 实现安全性的办法

接下来继续讨论前面提到的六个解决步骤。

第一步:可视化所有攻击方案　Java 安全性结构不是提出各种想得到的攻击方案,而是假定攻击方案的潜在目标集,类似于前面的攻击方案矩阵。

需要特别指出:Java 安全性模型涵盖了下列潜在目标:

- 内存
- 操作系统/程序状态
- 客户文件系统
- 网络

这些目标面对的是表 2.1 中列出的威胁:

- 破坏客户机上软件资源的完整性。
- 锁定/拒绝客户机上资源的使用。一般由病毒来完成。
- 窃走客户机上的信息。例如使用 UNIX 的 SENDMAIL 很容易做到这一点。
- 假冒客户计算机。可伪造 IP 地址来欺骗其它计算机和服务器。这种类型的攻击早在 Kevin Mitnick 作为黑客攻击计算机安全性倡导人之一的 Tsutumo Shimura 的个人计算机时就引起了世界关注。这个事件在纽约时报的畅销书《Takedown》(Tsutumo Shimura 著)中作了详细地讲述。

第二步:构造恶意行为基本集　Java 并没有得出恶意行为的基本集,只是提前使用了一个安全性热点基本集来构造安全机制以保障下列方面的安全:

- Java 语言结构和编译器。
- Java 编译过的类文件。
- Java 字节码验证器和解释器。
- Java 运行系统,包括类加载器、内存管理器和线程管理器。
- Java 外部环境,比如网页浏览器和接口机制。
- Java 小应用程序和小应用程序(构成可执行内容)被允许的自由度。

第三步:针对上述行为集设计安全性结构　构造一个隐含有禁止有害行为基本集机制的编程环境/计算机语言,从而禁止潜在有害行为,这就是 Java!

第四步:证明结构的安全性　在这一步包括从逻辑上(如果可能的话)以公理形式,证明语

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

言/环境免遭攻击方案攻击的安全性。

Java的安全结构尚未被公理地甚至是逻辑地证实是安全的。但是,Java确实把它的安全机制封装到了特殊的定义层中去了。每个安全性轨迹都可以被语言设计框架和Java语言程序与小应用程序的目标执行环境检查到。

第五步:将可执行内容限制在已被证实了的安全结构中 Java类文件检查器和字节码验证器就是为了达到这一目的而设计的。

第六步:保证安全性结构的可扩展性 这一步要求语言是可被设计的。应该将语言设计成为:一旦将来出现新的攻击方案就有相应的抵抗措施来处理,这种抵抗措施可以被翻新到基本安全结构中去。

封装在特殊的定义层中的安全性结构与Java安全管理器类结合在一起为Java提供了可扩展的安全性体系结构。

2.4.3 Java语言级的安全性

Java的第一级安全性是自身语言设计——句法和语义结构。下面让我们考察一下Java语言的安全性。

2.4.3.1 严格地面向对象

Java是彻底地面向对象的,每个元数据结构(当然派生的数据结构更是如此)都是完整的对象。每个封装的元数据类型都保证具有所有的理论上讲的安全性特点,这种面向对象的安全性优点渗透在Java程序的所有句法和语义中:

- 封装并隐藏私有声明后面的数据。
- 只通过公共方法控制访问数据结构。
- 可增加的和分级的程序结构复杂性。
- 不重载运算符。

2.4.3.2 Final类和方法

类和方法可被声明为final型的,这种类不允许被继承为子类,这种方法不允许以任何形式覆盖。这种声明可以防止对已验证过的可信任代码进行修改。

2.4.3.3 强类型和安全的类型强制

类型强制是静态的和动态的安全性检查,它保证编译时某个对象的类型严格地与最终运行时的类型相容,即使是通过类型强制做了转换的对象也是如此。类型强制可以防止恶意的类型伪装。

2.4.3.4 没有指针

这是安全性最强有力的保证,Java不使用指针。放弃指针的Java程序,没有任何部分是没有宿主的。每个数据结构和每个代码段都有一个句柄全程跟踪。

2.4.3.5 保证线程数据结构安全的语言句法

Java是多线程的,Java语言强制访问数据结构和对象的线程安全。本书在第13章"线程"

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

中使用了示例,应用程序代码详细地讲述了线程。

2.4.3.6　唯一的对象句柄

每个 Java 对象都有一个散列代码与之相关联。这就意味着 Java 程序的状态时刻都被完整地记录着。

2.4.4　编译过的 Java 代码的安全性

在编译阶段、隐含在 Java 语言句法和语义中的安全机制都要经过检查,包括私有和公共声明的一致性、类型安全和所有变量初始化,目的是保证各个变量都有一个已知的值。

2.4.4.1　类版本和类文件格式验证

每个编译过的类文件都是被验证过的,即需要证明它与当前已公布的正式的类文件格式一致。类文件需要经过结构检查和类文件格式各组件间信息一致性的检查。另外需要验证在类之间交叉引用(经过调用或变量赋值)时,公共和私有声明的一致性。

每个 Java 的类都被赋予了一个主版本号和一个次版本号,在同一个程序里需要检查类之间版本的匹配。

2.4.4.2　字节码验证

Java 的源类都被编译成了字节码。字节码验证器使得编译过的代码接受各种一致性和安全性的检查,对字节码的验证包括:

- 检查堆栈的上溢和下溢。
- 证实寄存器访问的合法性。
- 检查字节码参数的正确性。
- 分析用方法产生的字节码数据流以确保堆栈的完整性,确保方法传入和返回的对象与定义中声明的类型一致。

2.4.4.3　名字空间封装

Java 类是定义在包中的。Java 包名限定了 Java 的类名。包保证来自于网络上的代码与本地代码的区别。外来的类库不会恶变并冒用本地可信的类库,即使都拥有同样的名字也不会出现这种结果,这就可以保证在未验证的、偶然的本地类与外来类之间交互的安全。

2.4.4.4　推迟连接和绑定

推迟连接和绑定保证运行时刻的资源具有精确的布局,如堆栈和堆应尽可能地向后推迟。这种推迟的连接和绑定可以有效地阻止攻击方案对这些资源的攻击

2.4.5　Java 系统在运行时刻的安全性

运行时加载 Java 类的缺省机制是从本地宿主机上某个文件中取出被引用的类。其它加载类的做法,包括从网络上加载都要求相关联的类加载器。类加载器是标准的 Java Classloader 类的子类,它拥有执行所有一致性和安全性检查的方法,并使用这些方法对每一个新加载的类进行检查。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

由于安全性的原因，ClassLoader 不能对字节码做任何假定，字节码只能通过使用 Java 编辑器编译来生成。字节码也可以通过 C++ 编译器创建，经修改后成为 Java 字节码。这就意味着 ClassLoader 只有在字节码已经被验证过后才能工作。

ClassLoader 负责为下载的代码建立名字空间和为在下载的代码中引用的类转换名字。ClassLoader 强制了无限制的包的名字空间。

2.4.5.1 自动的无用内存单元收集和隐式的内存管理

在 C 和 C++ 里，程序员必须明确地分配内存、释放内存和跟踪所有指向已分配内存的指针。这对于维护来说是可怕的事情和主要的错误来源，这种错误是由于内存漏洞、悬挂着指针、空指针和分配操作与释放操作的不匹配所导致。

Java 取消了指针，这样就免去了程序员必须明确地管理内存的责任。Java 的内存分配和释放是自动的、严格结构化的和类型安全的。Java 使用无用单元收集来释放无用内存，而不是显式地释放。无用单元收集消除了与内存有关的错误和潜在的与内存有关的安全漏洞。人工分配内存和人工释放内存无法控制未经授权的复制、克隆和假冒信任对象和对数据一致性的攻击。

2.4.5.2 安全管理类

安全管理类是 Java 中可生长和可扩展的部分，是实现安全策略的核心部分，它为 Java 其它部分提供了安全性封装，这些部分包括类库和外部环境（比如支持 Java 的浏览器和本地方法）。Security Manager（安全管理）类不是单独使用的（安全检查缺省地产生安全异常），它是一个外壳类，它通过子类实现特定的安全策略集。

在其它特性中，安全管理类包含有判断安全检查是否在进行的方法，并且也检查下列项目：

- 防止附加的类加载器被安装。
- 是否有动态库被连接（用作本地码）。
- 类文件是否可读取。
- 是否可写入。
- 是否能够建立网络连接。
- 是否有监听连接的网络端口。
- 网络连接是否可以被接受。
- 是否可以访问特定的包。
- 是否可以向包中增加新类。
- 本地操作系统调用的安全性。

2.4.6 可执行代码的安全性

对 Java 程序构成威胁的因素主要来源于来自网络并在客户机上运行的 Java 代码。这种可传输 Java 程序的类叫做 Java 小应用程序类。从安全观点上讲 Java 小应用程序在语言框架上有明显的能力集和限制集。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

2.4.6.1　文件系统访问限制和网络访问限制

Java 对于从网络上加载小应用程序有下列限制：

- 它们不能读写本地文件系统中的文件。
- 它们不能创建、重新命名或拷贝本地文件系统中的文件和目录。
- 它们不能任意建立网络连接，除非是在它们的宿主机上。宿主机上应该有在 HTML 页的 URL 中的指定宿主域名（这个页里包含有小应用程序标志〈APPLET〉），或者在〈APPLET〉标志的 CODEBASE 参数中指定宿主名。宿主机的数字化 IP 地址不起作用。

上述针对本地文件系统访问的严格的限制集适用于在 Netscape Navigator 上运行的小应用程序，在 JDK 的 AppletViewer（小应用程序浏览器）上对此限制有小小的放松，它允许用户明确地定义一个特定的可被小应用程序访问的文件列表。

现在用户可以像将在第 16 章“JAR 档案文件”学习到的那样通过在浏览器中“签署”文件来克服对小应用程序的限制。这是可以使需要执行这种功能的小应用程序能够在维护安全框架时具有这种能力。

2.4.6.2　外部代码访问限制

小应用程序不能做下列事情：

- 不能通过 fork 系统或 exec 系统调用外部程序。
- 除了从小应用程序主线程上生长出来的自己的线程组以外，不能操作 Java 的任何线程组。

2.4.6.3　系统信息访问

小应用程序可以通过调用 System.getProperty(String key)来读取某些系统属性。在 Netscape 下小应用程序不限制对这些属性的访问。Sun 公司的 JDK 里的 AppletViewer 允许每个控制都可以访问这些属性。表 2.2 列出了各种关键变量的返回信息类型。

表 2.2　可用的系统变量

关键变量	返回的信息
java .version	Java 版本号
java.vendor	与 Java 供应商有关的字符串
java.vendor.url	供应商的 URL（统 -资源定位器）地址
java.class.version	Java 类版本号
os.name	操作系统名
os.arch	操作系统结构
file.separator	文件分隔符（比如/）
path.separator	路径分隔符（比如:）
line.separator	行分隔符

2.4.6.4　不可访问的系统信息

表 2.3 中列出的信息是在 Netscape 下小应用程序无法访问的信息。AppletViewer 浏览器和 HotJava 浏览器通过用户可控制的机制访问其中的一个或多个资源。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 2.3 限制小应用程序访问的系统变量

关键变量	返回的信息
java.home	Java 安装目录
java.class.path	Java 类的目录
user.name	用户的账号名
user.home	用户主目录
user.dir	用户的当前工作目录

2.4.6.5 从本地客户上加载的小应用程序

Java 系统(注意:只适用于 AppletViewer 浏览器)加载小应用程序有两种不同的方法。通过网络加载小应用程序和从本地文件系统加载小应用程序。加载小应用程序的方式取决于小应用程序的自由度。

如果通过网络加载小应用程序,需要使用 ClassLoader 并且要接受 ClassLoader 类和 SecurityManager 类的安全检查。如果小应用程序驻存在用户的 CLAASSPATH 环境变量中列出的目录里的文件系统中,那么就由文件系统加载器加载。

从安全的观点看本地加载小应用程序可以:

- 读和写本地的文件系统
- 加载客户上的库
- 在本地机上执行外部过程
- 退出 JVM(Java 虚拟机)
- 跳过字节码验证器的检查。

2.4.7 开放问题

前面我们从一般意义上考察了可执行内容的安全性问题,也在具体的 Java 框架中考察了可执行前面内容的安全性问题,现在我们开始讨论在目前版本的 Java 中未公开的某些文献的安全性。可以看到,Java 的安全机制对于某些类型的威胁可以取得百分之百的安全性。

下面的 Java 组件是安全性机的安全机制所在:

- 语言句法和语法
- 编译器和被编译过的类文件格式检查器及版本检查器
- 字节码验证器和翻译器。
- Java 运行系统,包括类加载器、安全管理器、内存管理和线程管理。
- Java 外部环境,比如 Java 网页浏览器和它们的接口机制。
- Java 小应用程序和被允许的小应用程序的自由度(它组成了可执行内容)。

由这些组件提供的安全性因下列不同程度的困难而以不同的方式遭到削弱或突破:

- 尽管 Java 句法提供了隐藏机制和控制机制,源代码的布局仍然可能出现混乱和意外的暴露。这种情况可能导致安全性遭到破坏,比如赋值线程和对象访问线程不安全或者本该声明为私有数据结构却被暴露成了公共资源。
- 正在使用的系统是与平台有关的、非 Java 语言的,比如是 C 语言的。保证该系统不遭到破坏的唯一办法是取得 Sun 公司的许可或将它与可供参考的实现方法比较,使用非 Java 语言编写的运行系统可能导致安全性遭到破坏,不使用 Sun 公司运行系统或经过验证的某个克隆版本,而有些人使用自制的或无名的版本,在这些版本里包含的是遭

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

到削弱的类加载器版本和遭到削弱的字节码验证其版本。

- Java 和非 Java 环境的接口，比如网页浏览器，可能遭到破坏。

在 Java 里不容易讲清楚的安全性问题(或者可执行内容的其它机制，因某种缘故)包括：

- 客户机上的 CPU 资源可能被窃取。某个用户可能发送一个小应用程序到你的计算机上使用你的 CPU 执行某些计算并将结果送回它自己的计算机上。
- 小应用程序可能包含有龌龊的或令人烦恼的内容(图像、声音或文本)。如果这样事情经常发生，用户就不得不阻止这种小应用程序。这样就需要在标准的 Java 类库中集成一个可由用户定义的内容过滤器。
- 小应用程序可以分配任意大小的内存空间。
- 一个小应用程序可以启动任意多数量的线程。
- 安全性遭到破坏可能引发互联网协议内在的缺陷，特别是那些在 Java 出现和可执行内容大量出现之前实现的部分。

处理安全问题总的方法是加密 Java 的小应用程序和增加数字化标记。ClassLoader、SecurityManager 、甚至字节码验证器都可以包含内置的解密方法和标记验证方法。

> **注意**：这些及其它与 Java 安全性有关的开放问题是时下有争论的问题，有专门的研讨，也有包括了安全性破坏方案的研究，特别是在在线的某些论坛中争论更是特别激烈。

2.5　Java API

Java 应用程序编程界面，即 API，是一套由 Sun 公司为使 Java 语言开发的类。它是为用户开发自己的类、小应用程序和应用程序而设计的。使用这套已经写好的类可以编写只有几行长度的 Java 应用程序，但如果使用 C 语言则需要数百行长度才能实现相当的功能。那么请问你选择哪一种语言。

在 JavaAPI 中类是被编组到几个包中的，每个包中可以有多个类和多个接口，进一步讲，每个条目还可以有多个属性，诸如域和/或方法。

尽管不了解太多关于 API 的知识也可以使用 Java 编写程序，但是每个由用户开发的类都依赖于 API 包中除 java.lang.Object 以外的至少一个类，java .lang.Object 是所有其它对象的超类。当用户开始开发更复杂的程序时，这些程序需要字符串、套接字和图形界面等，那么了解由 Sun 公司提供的对象以及这些对象的属性对于用户是十分有帮助的。

> **提示**：建议用户从 JavaSoft 下载 HTML 格式的核心 API，然后通读它，通过通读用户可以更好地体会 Java 语言是如何工作的。当用户通读各个包后，将会开始理解使用 Java 编程是多么容易和 Java 面向对象编程的功能是多么强大。

2.5.1　Java 核心 API

核心 API 就是 Java 1.1 提供的 API。这些包组成的对象保证可用，不管 Java 如何实现，只要这种实现支持 1.1 以上版本即可。Java 核心 API 包括：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- java .lang
- java .lang .reflect
- java .bean
- java.rmi、java.rmi.registry 和 java.rmi.server
- java.security、java.security.acl 和 java .security.interfaces
- java .io
- java .util
- java .util .zip
- java.net
- java .awt
- java .awt .image
- java .awt .peer
- java.awt .datatransfer
- java .awt .event
- java.applet
- java .sql
- java . text

注意:1.1 以下版本的这些包只能保证在支持 1.1API 的机器上可用。

2.5.1.1 java.lang

java.lang 包由 Java 语言的核心类组成。它不仅提供了基本的数据类型、比如字符和整型,而且还提供了通过 Throwable 类和 Error 类处理错误的手段。更进一步地,SecurityManager 类和 system 类还能使用户可以对 Java 运行系统进行一定程度的控制。

参见第 47 章"java.lang"可以更多地了解 java .lang。

2.5.1.2 java.io

java.io 是 Java 语言的标准输入/输出库。这个包提供了以多种方法建立和处理数据流的能力。它提供的类型简单的有 string ,复杂的有 Stream Tokenizer 。

2.5.1.3 java.util

java.util 包是由几个基本的实用类组成的,这些类不适合其它任何包。这些类是:

- Date 类,用于使用日期来管理和处理操作。
- Hashtable 类。
- 用于开发 ADT 的类,比如堆栈和向量。

参见第 46 章"数据结构及 Java 工具"可以更多地了解 java .util 包。

2.5.1.4 java.net

java.net 包就是那个使 Java 成为基于网络的语言的包。它提供了通过创建或连接套接字

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

或使用 URL 与远程资源通信的能力。使用这个包用户可以编写自己的 Telnet、Chat 或 FTP 客户和/或服务器等。

2.5.1.5　java.awt

java.net 包也就是通常说的 Java Abstract Window Toolkit（抽象窗口工具，简称 AWT）。它是由一些能为用户的小应用程序或单机应用程序创建丰富的、有吸引力的和实用的界面资源组成。AWT 包不仅包括用于管理的类如 GridBaglayout，而且还包括几个固定的交互工具，如按钮和文本框。更重要的是，Graphics 类提供了富有价值的图形能力，包括画几何形状的能力和显示图像的能力。

2.5.1.6　java.awt.image

java.awt.image 与 java.awt 包有十分亲密的关系。这个包由一些处理和操作来自网上图像的工具组成。

2.5.1.7　java.awt.peer

java.awt.peer 是一个界面包，这些界面用作代码和运行代码的机器的中介。使用这个包不需要做任何工作。

2.5.1.8　java.applet

java.applet 包是 API 中最小的包，但是由于包含了 Applet 类的缘故它又是最值得注意的包。Applet 类有许多有用的方法，它是小应用程序的基础，能够通过 AppletContext 界面提供有关小应用程序的环境信息。

2.5.1.9　1.1 版本中的包

下面的包是升级到 1.1 时加入的：

java.awt.datatransfer　java.awt.datatransfer 提供了用于数据转换的类。它包括用于裁剪板的新类和传送 Java 字符串的能力。

java.awt.event　在 1.0 以下版本中所有的事件使用的是一个名叫 java.awt.event 的类。这种做法已被证实为非常笨拙的和难以扩展的。为了解决这个问题，在 1.1 版中提供了 java.awt.event 包，使用户能够随心所欲地使用事件。

2.5.2　JavaBean API

JavaBean API 为软件组件定义了一个可移植的、平台中立的 API 集。JavaBean 组件可以插入到已有的组件结构中，比如：Microsoft 的 OLE/COM/Active 结构、openDoc 和 Netscape 的 LiveConnect。JavaBean 的优点是最终用户能够使用应用程序构造器加入 JavaBean 组件。一个按钮可以触发（即调用）一个在别的组件中画的棒状图，例如，一个实时的数据伺服组件可以用在别的组件中画的图来表示。

2.5.2.1　Java.rmi、java.rml.registry 和 java.rmi.server

Java.rmi、java.rml.registry 和 java.rmi.server 包提供了所有用户需要用于远程方法调用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

(RMI)的工具。通过 RMI 用户可以在远程计算机上(服务器)创建对象,这种对象使用起来和在本地上没有什么差别。

参见第 36 章"远程方法调用"可以更多地了解 RMI。

2.5.2.2 java.lang.reflect

java.lang.reflect 包提供了用户需要反射对象的工具。反射可以使用户能够检查运行时的对象以便确定构造者、方法和域。

参见第 48 章"反射"可以了解得更多。

2.5.2.3 java.security、java.security.acl 和 java.security.interfaces

java.security 包提供了为 Java 程序加密所需的必要工具。使用 java.security 包用户可以安全地在客户与服务器之间传入或传出数据。

参见第 34 章"Java 安全性深析"可以更多地了解 java.security 包。

2.5.2.4 java.sql

java.sql 包含 JDBC 即 Java 数据库连接。JDBC 可以用来访问相关的数据,如 Microsoft 的 SQL Server 或 Sybase SQL。

参见第 38 章到第 40 章可以更多地了解 JDBC。

> **注意:**所有关于 API 的资料都可以从 JavaSoft 的互联网站点上得到,它的网址是 http://www.javasoft.com。

2.5.3 JDK 1.2 新增加的内容

下面的包是升级到 1.2 时新增加的:

2.5.3.1 Java Enterprise API

Java Enterprise API 支持与企业数据库的连接和与合法应用程序的连接。使用这种 API,用户可以构造分布式的客户/服务器方式的 Java 小应用程序和应用程序。这些程序可以运行在企业内部任何操作系统或硬件平台上。

Java 数据库连接即 JDBC,是标准的 SQL 数据库访问接口,它提供了对广义的关系型数据库的统一访问。用户可能听说过 ODBC。Sun 公司消除了使 Java 成为当今各种计算工业的标准的绊脚石。

Java IDL 被按照语言中立的方式开发成 OMG 接口定义语言说明,用于指定对象和不同平台上的客户之间的接口。

Java RMI 是用 Java 编写的应用程序,在两端按对等方式或客户/服务器方式的远程方法调用。

2.5.3.2 Java Commerce API

Java Commerce API 为网上购买和财政管理提供安全保障。Java Wallet 是最初的组件,它

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

定义和使用了客户端的信用卡、借贷卡和电子现金交易的框架。想象一下，互联网将占有你的全部业余时间和全部金钱！

2.5.3.3　Java Server API

Java Server API 是一个可扩展的框架。它使用户能够开发互联网上的和企业网上的服务器端应用程序。Java 服务器 API 提供了对服务器和管理信息资源统一和一致的访问。对于开发者来说需要快速开发服务器端应用程序时就需要使用这种 API。

2.5.3.4　Java Media API

Java Media API 允许开发者和用户轻松灵活地运用网页上丰富的交互媒体。媒体框架有一个用于同步的时钟和播放声音、视频和 MID 的媒体播放器。Two - D 和 3D 提供了先进的图像模式。动画提供了运动和二维护对象变形。Java 共享提供了来自多个用户的应用程序共享，比如共享白板。最后，Telephony 把电话和计算机联系在一起。这个 API 可能是人们最乐于探究的 API。

2.5.3.5　Java Security API

Java Security API 提供了一个保证小应用程序和应用程序安全功能的框架。这种功能包括数字化标记、密码和授权。

2.5.3.6　Java Management API

Java Management API 提供了一个丰富的并可扩展的 Java 对象和 Java 方法集，这些方法和对象可用于建立基于互联网和城域网的企业网的小应用程序。它是与其它公司合作开发的，这些公司都是工业领袖一级的公司，像 AutoTrol 、Bay Networks 、GGS、BMC、Central Design System、Cisco Systems、Computer Associates、CompuWare、LandMark Technologies、Legato Systems、Novell、OpenVision、Platinum Technologies、Tivoli System 和 3com ，除此之外还有 SunSoft 。

2.5.4　Java 嵌入式 API

Java 嵌入式 API 指的是 Java 可以嵌入设备中的 API。这种设备不能够完全地支持 Java 核心 API。它包括基于 java.lang、java.util 和部分 java.io 的最小嵌入的 API。它定义了一系列特殊区域的扩充，比如网络和图形用户界面。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 3 章　安装并启动 JDK

本章内容：

3.1　为什么需要使用 Sun 公司的 Java 开发工具来编写 Java 程序

3.2　进一步了解 Java 既是编译的又是解释的

3.3　获取和安装 Sun 公司的 JDK

3.4　安装下载的 JDK

3.5　测试 Java 编译器和 JVM(Java 虚拟机)

3.6　在 Windows 3.1 平台上安装 IBM 的小应用程序开发工具

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

3.1　为什么需要使用 Sun 公司的 Java 开发工具来编写 Java 程序

本章将介绍 Java 开发工具和几个支持 Java 的浏览器。到本章结束时用户就已经安装了 Java 并编译和运行了第一个用户自己开发的 Java 应用程序。

Java 开发工具(JDK)是一个由 Sun 公司开发的免费软件包。这个软件包包括了所有编写 Java 程序和运行 Java 程序所需要的工具。它包括构成 Java 环境的所有基本组件,包括 Java 编译器、Java 解释器、小应用程序浏览器(不用打开兼容 Java 的网页浏览器就可以观察小应用程序)、以及一些用于开发 Java 应用程序的程序。JDK 可以使用户在编写程序时所要做的工作尽可能地减少。

如果没有免费的午餐,那么 JDK 就是一份比较好的快餐或小吃。许多程序员可能已经习惯了使用集成的开发环境,但它不是集成的开发环境,它没有完全包括开发 Java 程序的所有工具,JDK 的工具是命令行方式的,它没有漂亮的图形用户界面,如 Visual C++ 或 Borland C++,JDK 工具是在命令提示符下使用的(就像 Windows 95 和 Windows NT 里的 DOS 窗口那样)。包含源代码的文件就是用文本编辑器编辑的普通 ASCⅡ码文本文件,这种文本编辑器可以是 Notpad (Win32 系统提供)、vi(UNIX 系统提供)或者 BBEdit (Macintosh 提供)。

> **注意**:数量不断增加的集成开发环境(IDEs)可以从第三方公司获得,集成开发环境提供的各种特性可以使程序员的工作变得轻松。如果用户决定用集成开发环境作为自己 Java 开发工具,那么就应该找一个带有可以为 Java 代码加颜色的代码编辑器、带有工程文件管理器和速度较快的编译器的集成开发环境。许多大的开发公司都有 Java IDEs。Microsoft 公司的 Visual J++、Borland 公司的 JBuilder、Symantec 公司的 Cafe、IBM 公司的 Visual Age of Java、Metroworks 公司的 Codewarrior、Aysmetrix 公司的 SuperCede 就是几个可以得到的商用 Java 开发环境。每种开发环境都有其优点和缺点,如果用户打算做重要的 Java 开发,那么应该考察一下它们,看哪一个更适合用户的编程需要,从中选一个最合适的。即使用户打算使用像 Visual J++、Visual Cafe 或 Visual Age for Java 这样的集成开发环境,也应该了解一下 JDK,因为它是一个参考,可以把其它的开发环境拿来与它做比较。

3.2　进一步了解 Java 既是编译的又是解释的

C++ 编译器可以把高级的 C++ 程序编译成计算机微处理器能够理解的指令。这意味着对于不同的计算机平台需要分别地编译方能使程序正常运行。把 C++ 程序在不同的计算机上编译不是一件容易的事情。因为不同的计算机工作的方式不同,C++ 程序必须能够处理这些不同。这是至今处理多平台通用软件的一个重要问题。

Java 环境通过在编译器和计算机之间增加一个名为 Java 虚拟机(JVM)的中介人来解决这个问题。Java 编译器 Javac 不是直接面向某种计算机类型的计算机编译器。而是将文本文件中人可读懂的 Java 源代码编译成 Java 虚拟机能够理解的低级 Java 字节码。Java 虚拟机接管这些字节码并对它解释,这样 Java 程序就可以在装有 Java 虚拟机的计算机上运行了。唯一

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的面向平台的程序只有 Java 虚拟机本身，所有的支持 Java 小应用程序的网页浏览器都有内置的 Java 虚拟机。

Java 虚拟机概念提供了许多优点，最主要的一个是交叉平台兼容性。Java 程序员不必关心计算机如何处理特殊的任务，也不必关心如何编译不同版本的程序在不同的平台上运行。唯一需要用户关心的平台就是 Java 虚拟机。程序员有理由相信他们的程序可以在任何拥有 Java 虚拟机的平台上运行，比如 Windows 95，Solaris 和 Macintosh。

警告：即使是 Java，不同的平台上也有些差别。把程序尽可能地在各种平台上测试一下是一个好主意。

另一方面，像 Basic 这样的语言不是编译的。为了运行这样的程序必须拥有解释器，由解释器阅读每一行代码，分析程序的内容并转换成必要的机器代码来运行该程序。这种解释器的缺点是它需要处理许多事情所以速度不可避免地要慢一些。

因为 Java 能够被编译，这只是部分地解决了上述问题，但是可以将解释工作大大地减少。这种系统的主要缺点是解释代码的速度仍然比运行宿主机上的本地程序慢。Java 虚拟机必须把 Java 字节码的每个指令转化成能在系统上运行的相应指令，这也将导致处理 Java 程序的速度降低。

为了克服 Java 速度的限制，可以考虑使用即时编译器(JITs)。JIT 使得 Java 系统变得更加复杂，但是它能够将已经编译过的 Java 字节码进一步编译成给定机器上的本地代码，以此来加快运行速度，这对于用户来说是完全透明的。因为 JIT 是 Java 虚拟机的一部分，所以也是与平台有关的，但是它可以运行任何 Java 字节码，不管字节码来自于何种平台。使用 JIT 的 Java 程序可以取得与本地 C++ 程序相近的速度。

3.3 获取和安装 Sun 公司的 JDK

既然已经了解了一些关于 Java 和 JDK 的知识，那么用户可能就准备要实际地安装和使用它了。

如果用户还没有做这一步就请坐在计算机前，将计算机打开，在 JDK 光盘里有三个子目录：MACINTOSH、SOLARIS 和 WINDOWS。每个子目录中都包含完整的由 Sun 公司开发的 Java 开发工具即 JDK。表 3.1 列出了这张光盘中的 JDK 目录下三个子目录中的内容。

表 3.1 光盘中 JDK 文件夹中的内容

目 录	内 容
MACINTOSH	包含 Macintosh 平台上的 JDK，适用于 68K 和 Power PC
SOLARIS	包含两个子目录，一个是适用于 SPARC Solaris 的 JDK，另一个是适用于 x86 Solaris 的 JDK
WINDOWS	包含适合于 x86 上 32 位 Windows 操作系统的 JDK，即 Windows 95 和 Windows NT 操作系统上的 JDK

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:用户也可以使用网络浏览器从互联网上得到 JDK。如果用户想下载它请参考本章后面的"下载 JDK"部分。

读者是否使用过这三种平台的任何一种?读者可能是幸运的也可能是不幸的。其它的平台上也有许多 JDK,但是读者需要在国际互联网上查找。表 3.1 中的三种是由 Sun 公司支持的,而其它的 JDK 不是由 Sun 公司支持的。有一部分是用在下列系统中的:linux、DEC Alpha、Amiga、OS/2 等等。查找与这些发行物有关的信息的最好地方是 Sun 公司的 http://www.javasoft.com/products/jdk/1.2/。

现在看一下如何在 32 位 Windows 操作系统上安装光盘里的 JDK。安装是很容易的,但是在安装 JDK 之前应该熟悉 Windows 环境和 DOS 环境。

3.3.1　从光盘安装在 Windows 95 和 Windows NT 上的 JDK

第 1 步　删除 JDK 的老版本 一般情况下在读者的机器上可能没有老版本的 Java 开发工具。如果读者在 JDK 目录下有其它的 Java 源代码文件(读者编写的或者从别人那里得到的)。那么请读者将这些文件移动到一个新目录中去。以免在删除老版本的 JDK 时将它们错误地删除了。现在读者可以使用 Windows 的浏览器或源管理器删除整个 Java 目录了。

第 2 步　打开 JDK 包 删除完老版本 JDK 后就可以通过 JDK 文件自展将 JDK 包打开。应该在 C 盘根目录下创建一个名为 JDK 1.2 的目录,在这个目录下打开 JDK 包。如果希望在其它目录下安装 JDK,可以在相应的目录下展开 JDK 包。打开 JDK 包的操作将创建 Java 父目录和 2 版必需的子目录。

如果浏览一下 JDK 包安装的文件可发现 lib 目录下和 jre \ lib 目录下有几个以.jar 为后缀的文件。jar 文件包含成功地运行 Java 虚拟机必须的运行时刻 API 类。注意:在 1.2 以前的 JDK 版本中安装程序创建名为 lib \ classes.zip 文件,而不是各种.jar 文件。**不要展开 CLASSES.ZIP 文件。**

第 3 步　更新环境变量 当包展开以后,应将 JAVA \ BIN 目录加在路径中,只要用文本编辑器打开 AUTOEXEC.BAT 文件,在路径语句中加上 JAVA \ BIN 即可。

如果用户设置了 CLASSPATH 环境变量,那么可能需要修改它。例如,可能需要把 CLASSPATH 项指向 jdk1.2 \ jre \ lib \ rt.jar 文件。最容易的做法是用文本编辑器打开 AUTOEXEC.BAT 文件修改 CLASSPATH 环境变量,或者是用 setup 程序来修改这个变量。

修改完毕 AUTOEXEC.BAT 文件后,将它存储在硬盘根目录下并重新启动机器,这样就可以使修改后的环境变量发挥作用了。

接下来将介绍 x86 和 SPARC Solaris UNIX 系统上的 JDK 安装。这个安装过程很像某些其它的 UNIX 操作系统安装。有关如何获取 UNIX 系统上的 JDK(如 linux)的信息详见第 56 章"Java 资源"。

3.3.2　从光盘安装在 x86 和 SPARC Solaris 操作系统上的 JDK

在 32 位 Windows 系统上安装 JDK 是非常容易的,但是要求在安装之前应该熟悉 Windows 环境和 DOS 环境。

第 1 步将目录拷贝到驱动器内　将适当的目录(或者是 x86 目录,或者是 Sparc Solaris 目

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

录)拷贝到硬盘驱动器内。根据文件系统的结构和系统的特权级,可能希望拷贝该目录到公共区域内,比如/usr /local /或者是主目录。把光盘上的 Solaris 目录拷贝到主目录的命令是:

```
> cp -r sparc ~/
```

第 2 步设置环境变量 CLASSPATH 是一个环境变量,它将路径定义到了 rt.jar 文件中。JDK 带的许多工具都使用 CLASSPAH 变量来定位文件,所以正确设置它是非常重要的。可以通过在命令提示符下输入下列内容来设置 CLASSPATH 变量:

```
% setenv CLASSPATH.:/user /local /jdk1.2/jre /lib/rt .jar
```

或者可以把这个文本行放置在 login 文件或 profile 文件中,这样可以在每次登录时调用它:

```
setenv CLASSPATH.:/user/local/jdk1.2/jre /lib /rt.jar
```

> **注意**:如果使用早于 1.2 的 JDK 版本,需要用 lib/class.zip 替换 jre /lib /rt.jar,在本书的所有例子中都需要如此替换。

3.3.3 下载 JDK

除了从光盘中获得 JDK 以外还可以从国际互联网上下载 JDK。从国际互联网上下载肯定可以得到最新版本的 JDK。

下载 JDK 需要什么? 第一个需要就是有一个与国际互联网联接着的计算机来运行网页浏览器。不需要什么特殊的浏览器,只要有 Netscape Navigator 浏览器即可做完这些例子。

第二个需要是计算机的硬盘空间,用于存储下载的 JDK 文件。表 3.2 列出了在各种平台上下载 JDK 需要的磁盘空间和解压缩后需要的磁盘空间。

表 3.2 JDK1.1 需要的磁盘空间

平台	压缩状态所需的磁盘空间	解压缩后需要的磁盘空间
Solaris	13.7MB	16.5MB
Windows	5.77MB	12.1MB

开始下载 如果有足够的磁盘空间和浏览器就可以开始下载 JDK 了!

1.建立网络连接和启动网页浏览器。如果读者不知如何做,可向你的系统管理员或内行的朋友咨询、查阅手册或使用网页的书,比如 Que 的《国际互联网使用大全》。

2.为浏览器指明下载 JDK 的站点

http://www.Javasoft.com/products/jdk/1.2/

3.在"Down JDK Software"弹出式菜单中列有可从 Sun 公司得到的用于各种操作系统上的 JDK。从这个弹出式菜单中选出要使用的操作系统。

4.单击弹出式菜单下方的"Download Software"按钮。

5.这时遇到一个有许多 JDK 分发限制的页。读完它,如果同意这些限制,就单击"YES"按钮进入下载页面。

6.这一页上是一些可下载 JDK 的站点。选择一个离用户最近的站点,单击该站点就可以开始下载 JDK 了。

JDK 是一个很大的文件,下载它需要一定的时间。时间的长短取决于连接的速度、使用

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

FTP 服务器的人数、所处的位置、下载文件的时间、正访问该服站点的人数和运气好坏以及许多难以预言的因素。如果文件传输的速度太慢可以另找一个时间再连接。时间的选择应考虑所在时区的因素，当然好的连接机会是各种各样的，但是这些机会还是与上述提到的因素有关。

3.4　安装下载的 JDK

一旦硬盘上有了下载来的安装文件后，就可以将其安装在机器上并开始编写程序了。每一个平台都有它自己标准的安装过程，而且 JDK 安装程序做得非常好，只要按照它的提示做下去就可以顺利而轻松地将 JDK 1.2 安装成功。

3.4.1　在 Solaris x86 平台和 SPARC 平台上安装

对于 Solaris，Sun 公司发布的是一个标准的自展开外壳描述文件(文件以.sh 为扩展名)；文件的名字指示了它的版本。

警告：在开始安装新版本之前应使用 tar 或其它标准的系统备份过程将前一个版本的 JDK 备份。一般人都不希望丢掉在前一版本下所做的工作，另外做了备份也可以防止新版本出现错误。

在 Solaris 机器上安装 JDK 有两种方法。一种方法是安装在用户的主目录里仅供用户使用，另一种方法是安装在公共的 bin 目录里，比如/usr/local/bin/，这样系统中的所有用户都可以使用它。两种安装方法的过程都是一样的。

1.选择安装目录。假设安装的位置是/usr1/JDK1.2。如果选用了其它目录可用相应的目录名代替 usr1。例如，如果选择安装在用户的主目录，可以用～或 $ HOME 来代替 usr1。

2.验证安装目录的写入许可。使用下面的命令检查当前的权限：

ls －ld /usr1

ls 命令的选项指定了一个长列，这个列中包括所有者信息和权限信息。用户也可以指定该目录中不列出的内容，这是缺省的。更详细的关于 ls 命令的信息可参考用户的系统手册。

该命令的输出应该是像下面的样子：

drwxr －xr －x root other 512 feb 18 21:34 /usr1

在这种情况下目录的拥有者是根(系统管理员)并且既不允许组写入也不允许普通用户写入该目录。如果用户遇到这种情况并且又不是根，那么就需要请系统管理员来帮助用户在这个目录里安装 JDK。

3.将 JDK 发布文件移动或拷贝到/usr1 目录。

4.通过输入一个圆点、一个空格和 jdk.sh 文件名展开 JDK(比如 jdk1.2 －solaris2－sparc.sh)。

> . jdk1.2－solaris2－sparc.sh

这一命令将执行外壳描述文件，它将自动地将所需的文件解压缩到该目录中。

5.验证下列目录是否在/usr1 目录中被建立。

jdk1.2

jdk1.2/bin

jdk1.2/classes

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

jdk1.2/demo
jdk1.2/lib
jdk1.2/src

6.设置 PATH 环境变量。对于 C 外壳和它的派生程序,使用:

setenv PATH $ PATH:/usr1/jdk 1.2/bin

对于 Korn 外壳和它的派生程序,使用:

PATH $ PATH: /usr1/jdk1.2/bin
export PATH

7.设置 CLASSPATH 环境变量。对于 C 外壳和它的派生程序,使用:

setenv CLASSPATH /usr1/jdk1.2/jre/lib/rt.jar

对于 Korn 外壳和它的派生程序,使用:

CLASSPATH = CLASSPATH
/usr1/jdk1.2/jre/lib/rt.jar
export PATH

提示:与其每次从命令行下设置这些变量,不如在外壳资源文件,~shrc、cshrc、profile 等中加上该命令行。如果用户是系统管理员并且要把 JDK 当作网络开发工具来安装,那么可能希望将这些参数加到缺省的设置文件中。

3.4.2 在 Windows 操作系统上安装

用户一般是在 Windows 95 或 Windows NT 上运行 Java。如果要在 Windows 3.1 上安装 JDK,可以阅读本章后面的"在 Windows 3.1 上安装小应用程序开发工具"部分。

安装 JDK 是一个很简单的过程,但是用户应该了解 Windows 环境和 DOS 环境。Windows 环境里的 JDK 提供了标准的安装窗口格式;文件名显示了它的版本。

1.选择安装目录

这些指令假设安装的位置是 C:/JDK1.2。如果选择了不同的主目录,可以直接把适当的路径(并且可能需要改变驱动器符)追加上去。如果说希望安装在 E:/TOOL/JAVA/目录下,可以在所有的指令中用 c:/tools 替换 C:。

警告:可以使用 Windows 95 或 Windows NT 的资源管理器(Explorer)将 JAVA 目录改名(例如改成 OLDJAVA)。如果因为某种原因导致安装失败,用户可以把前一版从 OLDJAVA 目录中恢复出来。另外如果安装成功了还可以把附加的文件,比如文档,在删除老版本之前将其移到新版本中。

2.如果计划安装在网络驱动器上应确保拥有对相应目录的写入权限。

3.通过运行自展开程序将 JDK 展开(在浏览器里或文件管理器双击该文件的图标)。

4.验证下列子目录在 C 驱动器下已经被创建:

C: \ JDK1.2
C: \ JDK1.2 \ BIN
C: \ JDK1.2 \ CLASSES

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

C: \ JDK1.2 \ DEMO

C: \ JDK1.2 \ LIB

提示:对于 4.0 及以上版本的 Windows NT 。用户可以跳过第 5 步、第 6 步和第 7 步及从属性表中设置 CLASSPATH。因为用户不必重新启动机器,但是可能必须关闭所有 DOS 窗口,然后重新打开才能使新的变量生效。

5.在 autoexec.bat 文件里的 PATH 语句中增加 C: \ JDK1.2 \ BIN 。

set PATH = c: \ windows;c: \ dos;……c: \ java \ bin

6.在 autoexec.bat 文件中设置 CLASSPATH 环境变量:

set CLASSPATH = c: \ windows;c: \ dos;…c: \ java \ bin

7.重新启动计算机使新的环境变量生效。

3.4.3 在 Macintosh 上安装

用于 Macintosh 的 JDK 是一个二进制——十六进制的文档(带有 HQX.SIT 后缀的文件)。文件版本显示在文件名中。

警告:在安装新版本之前,一定要把当前版本的 JDK 保存好。用户不希望丢失自己前面做过的任何工作,并且可以在新安装的版本出现错误时仍可以拥有前一个版本。

在 Macintosh 上安装 JDK 的步骤如下:

1. 按照本章前面介绍过的方法下载完 Mac JDK 1.2 以后,用户应该拥有一个名为 MacJDK.SEA 的安装程序。双击这个安装程序就可以激活一个非常标准的 Macintosh 安装程序对话框。

警告:Macintosh 允许用户使用在 UNIX 中不允许使用的方式命名目录和文件。在 UNIX 中不能在文件名和目录中使用斜杠(/)。这可能引发问题,当 JDK 在试图定位文件时使用了混合的 UNIX/MAC 方法跟踪路径,这时目录名中的斜杠将被解释为改变目录。

UNIX 命令也有问题,它可以包括空格。用户应当遵守开发者使用的 UNIX 文件和目录命名约定。这就意味着不能使用空格、斜杠、统配符和其它标点符号来命名文件和目录。用户可以使用圆点,可以使用任意长度的文件名(只要不超过 32 个字符就行)。

例如下面的文件名在 Macintosh 中是很好的文件名,但是在 UNIX 中却不能使用:

/../.../Stuff \ / \ /..java

要想既能用于 UNIX 也能用于 Macintosh,那么文件名应该是像下面的例子那样:

Stuff.java

2.在安装程序对话框的左下角用户可以指定想要安装的位置。在选择完驱动器和目录后,可以单击"Install"按钮开始安装或按下"Return"键返回安装程序。不管在安装程序中指定哪个位置,Macintosh JDK 都将被安装在名为 MCJDK 目录下。缺省的安装位置是启动盘根目录下的第一级子目录中。

现在用户的硬盘驱动器中已经有了 JDK 的拷贝。它包括两个基本的程序:Java 编译器和小应用程序浏览器 AppletViewer。用户可以进入到下一步的(很有意思的)Java 开发部分。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

3.5 测试 Java 编译器和 JVM(Java 虚拟机)

现在用户可以写一个比较小的 Java 应用程序来测试一下 JDK 的安装。

3.5.1 创建新的 Java 工程

在硬盘驱动器上创建一个新目录用以存储用户的工程。假设在 JDK 目录之外建立一个名为 PROJECTS 的工程目录,这样可以在安装新版本 JDK 时不必将其移动。在这个目录中创建另一个名为 HELLOWORLD 的目录。

现在使用用户喜欢的文本编辑器(比如写字板 NotPad 、vi、emacs、SimpleText 或其它的文本编辑器)创建一个名为 HelloWorld.java 的文件(仔细检查大小写,Java 是大小写敏感的),然后输入下列内容:

```
public class HelloWorld {
  public static void main (String [ ] args){
      System.out .print ln ("Hello, World !");
  }
}
```

现在先不要关心具体的句法;把它输进去保存好并从文本编辑器中退出来。要确保它是标准的 ASCⅡ码文本文件。

3.5.2 运行 Unix 或 Windows 的 Java 应用程序

如果机器使用的是 Unix 操作系统或者 Windows 操作系统,那么就可以在命令(DOS)提示符下输入下列内容:

```
javac HelloWorld.java
```

机器应该暂停一会儿,然后返回到提示符下。

在 DOS 窗口中查看一下目录列表,确保其中有下列文件:

```
> dir
HelloWorld.class HelloWorld.Java
```

在 UNIX 系统中用 ls 命令查看目录的列表,确保其中有下列文件:

```
> ls
HelloWorld.class HelloWorld.java
```

如果出现错误,请仔细检查 HelloWorld.java 代码,确保它与书中前面的代码完全一致。

如果出现"javac 找不到"这种错误,是因为 PATH 变量中没有设置 JAVA/BIN 目录。那么就应该回到前面检查一下 JDK 安装。

如果未出现任何错误,就可以开始运行用户的第一个 Java 程序了。在命令提示符下输入下列内容:

```
> java HelloWorld
```

这时用户将会看到下列结果:Hello, World!

得到这一结果是一件值得庆贺的事情。这样用户就成功地运行了自己编写的第一个 Java 应用程序,更重要的是用户正确地安装了 JDK。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

如果没有看到“Hello, World”这一显示结果，那么说明 JDK 安装出现了错误。请检查 CLASSPATH 变量，确保其指向当前的工作目录(圆点“.”)和 rt.jar 文件。确保输入的文件名是正确的，请记住 Java 是大小写敏感的。如果这些都没有问题，那么就需要重新安装 JDK 了。

3.5.3　在 Macintosh 上运行 Java 应用程序

在 Macintosh 上编译和运行 Java 应用程序稍微有点困难，因为它没有命令提示符。

1.在 Macintosh 上打开 HELLOWORLD 文件夹可以看到 HelloWorld.java 文件。

2.然后打开 MACJDK 文件夹可以看到 Java 编译器的图标(有点像是一只胸前带有“C”字的鸭子)。将 HelloWorld.java 文件拖进 Java 编译器图标中。这时 Java 编译器启动并开始编译 HelloWorld 程序。当编译结束时在 HELLOWORLD 文件夹中将产生一个名为 HelloWorld.class 的文件。

3.如果出现编译错误，请检查 HelloWorld.java 的源代码，确保它和书中前面的代码完全一致。

4.如果没有出现编译错误可以双击 HelloWorld.class 文件。这时 java 应用程序运行器启动，开始加载 HelloWorld 程序并运行该程序。在屏幕上有一个名为 stdout 的窗口出现，窗口里面显示“Hello, World”。

如果是这样的话，祝贺你。你已经成功地安装了 JDK 并成功地运行了你的第一个 Java 程序：

如果没有看到“Hello, World!”那么说明安装有错误。检查并确保正在运行的是 System 7，检查是否完整地安装了 JDK，生成的文件名和类名是否匹配，请记住 Java 是大小写敏感的。如果上述都没有错误，那么就需要重新安装 JDK。

> **注意**：Macintosh Java 运行器应用程序的作者很聪明地将 Quit 命令从 Apple 菜单中隐藏起来了。他们为什么这样做至今尚不清楚。如果用户想在 Java 运行器运行完程序后释放 Java 运行器所占的内存，可以选择 Apple 菜单、单击 Java Runtime 然后单击 Quit。这样做有点不像 MAC 风格，只是因为 Macintosh 没有命令行。

像普通的 Macintosh 程序一样用户可以通过单击 Q 命令退出应用程序。

3.6　在 Windows 3.1 平台上安装 IBM 的小应用程序开发工具

为什么 Windows 3.1 平台上的 JDK 不是 Sun 公司开发的呢？大量的技术问题导致 JDK 在 Windows 3.1 上的实现比较困难，并且由于 Windows 95 的发行使得 Windows 3.1 显得有些过时，像是一个垂死的平台，所以 Sun 公司决定不再支持它。Java 需要长文件名支持，比如“.Java”和“.Class”。而 Windows 3.1 的八个字符的文件名和三个字符的扩展名命名系统无法支持 Java 的长文件名。更困难的是：Java 是多线程的语言，这就意味着它可以同时运行多个进程，但是 Windows 3.1 不支持多线程。为了在 Windows 3.1 上支持 Java，不止一个组织从事了将 JDK 移植到 Windows 3.1 平台上的工作，其中最成功的是 IBM 公司的 ADK。

通过使用 IBM 公司的 ADK 版本，Windows 3.1 用户可以有办法开发 Java 小应用程序和应用程序，而不必升级到 Windows 95 或 Windows NT。它包含了大量用于解决前面提到的问题的程序以及 JDK 工具。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下载 ADK

为了进入 ADK 的网页，首先要打开网页浏览器并指定连接地址为 http://www.alphworks.ibm.com/。这个网页包含大量的 IBM 正在开发的项目。为了进入 ADK 网页，需要在弹出式菜单的"Select"部分选择"ADK For Win3.1"条目。

为了安装和使用 JDK 的特性，Java 需要三个组件：ADK、Windows 的 32 位扩展 Win32s 和 WinG 图形扩展。

为了下载后两个组件请连接 ftp://ftp.microsoft.com/softlib/mslfiles/并获得下述两个文件：

pw1118.exe

wing10.exe

WinG 扩展的文件名是 Wing10.exe，它的容量大约为 830KB。

Win32s 的文件名是 Pw1118.exe，容量大约为 2.4MB。在安装 ADK 之前需要得到并安装它们。

为了安装这两个系统增强组件需要为它们建立两个目录并将.exe 文件分别放在其中。可以在 DOS 提示符下执行.exe 文件，也可以在程序管理器的文件菜单里的 Run 命令中执行.exe文件。如果把 wing10.exe 文件存放在 C 驱动器的名为 wingtemp 的目录下，那么 DOS 提示符下的命令应该是下面这个样子：

C:\wingtemp\ > wing10.exe

这个命令将把所有要安装的文件全部解压。两个文件都要解出一大堆文件，其中各有一个可执行文件，名为 setup。解完压缩以后再在 DOS 提示符下或使用 File 菜单中的 Run 命令运行 setup 程序。setup 程序提示一些信息并安装所有必需的文件。当做完安装工作以后就可以把这两个放置安装程序的临时目录删除。

当安装完 WinG 和 Win32s 以后就可以着手安装 ADK 本身了。首先需要阅读 http://www.alphaWorks.ibm.com/ADK 上的 ADK 的许可协议。

在这一页的底部有一个标有"I Agree"的按钮。如果阅读完了许可协议并同意其中的所有条款就可以按下这个按钮，这样就可以进入 ADK 安装程序的下载页了。实际的 ADK 文件是相当大的，大约 4MB，下载它需要较长的时间，特别是通过调制解调器下载。

一旦得到了 ADK 安装程序就可以在 Windows 程序管理器的 Run 菜单中执行它。安装程序将提问安装目录(比如 C:\java\)，得到确认后它就开始安装所有用于启动 ADK 的文件。

当 ADK 安装完毕后，它建立一个程序组，如表 3.3 所示。

表 3.3 ADK 程序组中的文件

文件名	描述
Configure AppletViewer	运行 AppletViewer 并显示许可文档
ADK WRI	ADK 用户指南 - ADK 解释信息
ADK File	文件管理器类型应用程序，用长文件名代替 Windows 3.1 的 8.3 文件名(8 字节文件名，3 字节扩展名)来操作文件
ADK Edit	集成了 ADK 工具的编辑器，不必在程序间切换即可以用 Java 代码工作
ADK Console	ADK 内核，它是运行所有基于 Java 环境的工具程序，比如 AppletViewer 和 Javac

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

为了安装 ADK，应运行“Configure AppletViewer”程序，同意许可协议并按照指令设置 AppletViewer，然后关闭应用程序。

为了测试安装是否成功可以执行下列几个步骤：

1. 启动“ADK Console”程序。

2. 从 Tools 菜单中选择 AppletViewer。

3. 将 C: \ java \ demo \ Animator \ 输入到 Working Directory field(或安装有 ADK 的目录)。

4. 在 Command Option field 中输入 Examplel.html。

5. 按下 OK 按钮。

这时 Animator 小应用程序开始启动并有一只鸭子在屏幕上跳舞。如果是这样，说明 ADK 安装成功，用户可以在 Windows 3.1 平台上开发 Java 程序了。如果不是这样，请检查 Working Directory Field 中输入的是否是 Animator 小应用程序所在的目录以及 Examplel.Html 文件是否在那个目录中。如果这方面没有问题，那么就需要回到前面重新安装 ADK。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第4章 JDK工具

本章内容：

4.1 JDK工具参考
4.2 AppletViewer(小应用程序浏览器)
4.3 java,Java解释器
4.4 javac,Java编译器
4.5 javap,Java反编译器
4.6 javah C头文件和承接模块文件的创建器
4.7 javadoc工具(文档生成器)
4.8 jdb,Java调试器
4.9 CLASSPATH环境变量
4.10 Macintosh问题
4.11 Java编译器
4.12 Javah:C头文件产生器

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

4.1　JDK 工具参考

本章将介绍 Java 开发工具中的所有工具。将介绍每个工具是做什么的，所有与工具有关的选项和它们引用的环境变量。如果用户正要开始用 Java 编写程序，那么可以把这一些内容作为工具的介绍来学习。如果用户是一位铁杆黑客，那么更应该把它作为工具参考来使用，这样可以节省时间和机时。无论是什么水平 Java 程序员阅读本章都能够得到启发，了解 JDK 工具能做什么和掌握如何使用它们的方法。

4.2　AppletViewer(小应用程序浏览器)

小应用程序是用 Java 编写的嵌入到 HTML 文件，比如网页中运行的程序。在许多情况下它们是不能够独立运行的。小应用程序浏览器工具是一个小程序，用于在不启动网页浏览器情况下运行小应用程序。在开发小应用程序时小应用程序浏览器是一个很快和很容易的测试方法。

用下面的命令就可以调用小应用程序浏览器：

AppletViewer [选项] URLS……

命令行中的 URLS 是 HTML 文件的统一资源定位器地址比如(http://www.javasoft.com/index.HTML)，HTML 文件中包含有小应用程序标志。如果用户当前在某个含有引用了小应用程序的 HTML 文件的目录里，用户可以通过输入包含有小应用程序标志的 HTML 文件的名字来调用小应用程序浏览器。下面是小应用程序浏览器可以使用的选项：

选项	描述
-debug	在 Java 反编译器 jdb 中启动小应用程序浏览器，这样可以在 HTML 文档中调试小应用程序。

小应用程序浏览器窗口中有一个小应用程序菜单，用这个菜单可以为小应用程序浏览器设置许多不同的功能。它的菜单选项如下所示：

- Restart　使用当前设置重新启动小应用程序。
- Reload　重新加载小应用程序。当类文件发生变化时需要重新加载。
- Stop　调用小应用程序的 Stop()方法，停止小应用程序的运行。注意：此时小应用程序并没有析出。
- Save　存储小应用程序的序列化状态。
- Start　启动小应用程序。当使用过 Spot 选项以后这个选项就显得有用了。如果小应用程序没有停止，它就没有动作。
- Clone　克隆(复制)当前的小应用程序，使用同样的设置创建另外一个小应用程序浏览器实例。
- Tag　在 HTML 文件中用于指示浏览器运行小应用程序和从 HTML 文件(见图 4.1)中传出参数的标志。
- Info　显示有关小应用程序的特殊信息，这些信息设置在小应用程序的程序中(见图 4.2)。
- Edit　这个选项不做任何操作，自从第一个测试版开始它一直是灰的。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- Print　　将小应用程序的 PrintGraphics 送往打印机。
- Properties　　显示小应用程序浏览器的安全属性。这些属性设置后允许用户为网络环境配置小应用程序浏览器，包括防火墙代理、HTTP 代理、使用有关的代理服务器和代理端口框。网络访问框允许用户选择小应用程序浏览器允许的网络访问类型。选项分别是 No Network Access、Applet Host（缺省值）和 Unrestricted。Class Access（类访问）框允许用户选择访问方式 – Restricted（受限制的）和 Unrestricted（不受限制的），使用 Unrestricted 选项可以查看其它的类，如图 4.3 所示。
- Close　　关闭小应用程序浏览器窗口并终止小应用程序。
- Quit　　关闭小应用程序浏览器窗口并终止小应用程序。

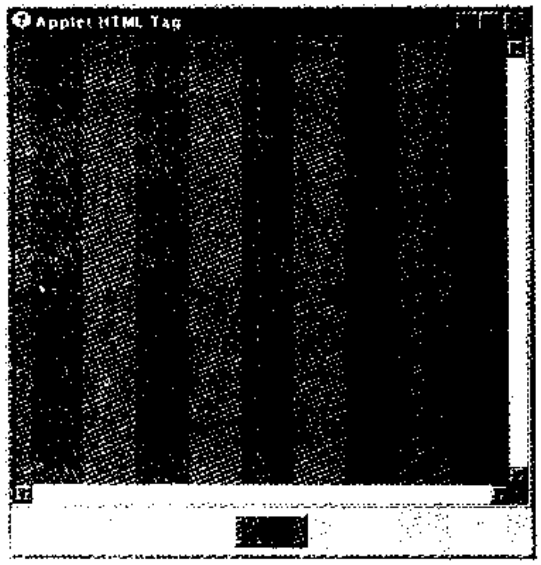


图 4.1　A 小应用程序浏览器的标志窗口

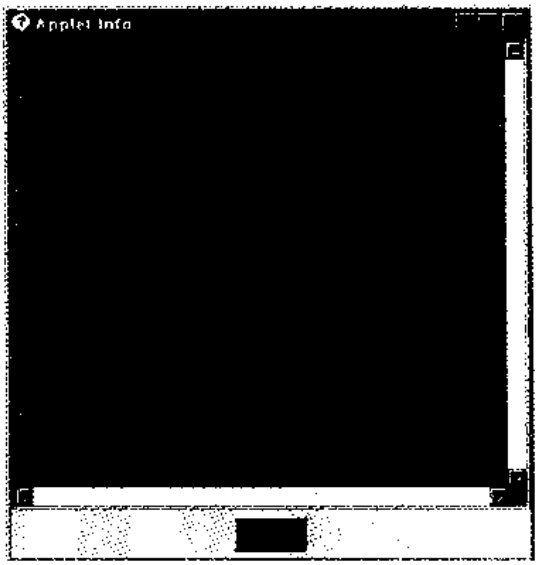


图 4.2　小应用程序浏览器的小应用程序信息窗口

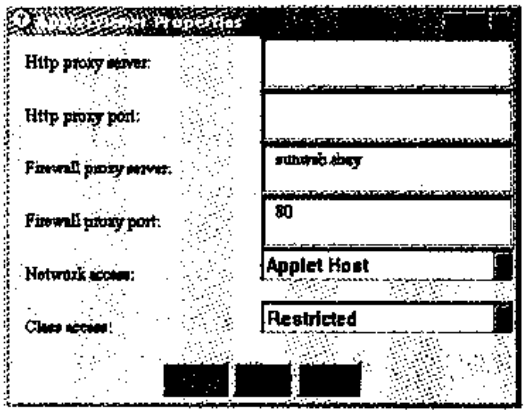


图 4.3　小应用程序浏览器的属性窗口

4.3　java，Java 解释器

Java 解释器用于运行编译过的 Java 应用程序。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解释器的句法是：

java［选项］类名

类名中只包括类的名字，不包括扩展名(.class)。Java 解释器的选项在表 4.1 中列出。

表 4.1　Java 解释器选项

选项	描述
-help	显示所有的选项
-version	显示编译源代码的 JDK 的版本
-v (同于 -verbose)	显示被加载的所有类(与在 javac 工具中的功能一致)
-cs (同于 -checksource)	检查源代码是否比类文件新(未被编译过)，如果比类文件新就编译新版本的源代码
-debug	使用远程 Java 文件，该文件要在后面用 jdb 工具调试。解释器为用户产生一个口令，这个口令在 jdb 口令选项中使用(参考本章后面的"jdb——Java 调试器"部分)
-prof	将配置信息输出到 \ JAVA.PROF 文件中
-classpath dirs	Java 在指定的 DIRS 目录中寻找类文件。对于多目录情况使用冒号(在 Unix 中)或分号(在 DOS 中)将每个目录分开。例如在 DOS 机器上，类路径可能像下面这样设置： CLASSPATH = .;c: \ users \ dac \ classes;c: \ tools \ java \ classes
-noasynegc	关闭异步的无用单元集
-verify	验证所有被加载的类
-verifyremote	验证引入的类或继承的类。这是缺省设置
-noverify	关闭类验证
-mx val	为由 val 指定的值设置的最大的 java 堆空间。最小的堆空间是 1KB(-mx 1k)，缺省值是 16MB(-mx 16m)。(字符 m 代表兆字节，字符 k 代表千字节)
-ms val	将初始的 Java 堆空间设置为由 val 指定的值。最小的堆空间是 1KB(-mx 1k)，缺省值是 1MB(-mx 1m)。(m 代表兆字节，k 代表千字节)
-ss val	将 C 过程的堆栈空间设置成由 val 指定的值。该堆栈空间必须大于 1KB(-ss 1k)(字符 k 代表千字节)
-oss val	将 Java 过程的堆栈空间设置成由 val 指定的值(m 代表兆字节，k 代表千字节)

4.4　javac，Java 编译器

javac 程序是将.java 文件转换成可由解释器执行的类文件的工具。表 4.2 列出了 java 编译器的选项。

表 4.2　java 编译器选项

选　项	描　述
-O	显示当前 JDK 的版本
-classpath < path >	覆盖缺省的 CLASSPATH 环境变量并指定新的类搜索路径。一定要包含库类，比如 jdk1.2 \ jre \ rt.jar
-d < directory >	指定存放生成的类文件的目录。注意该目录要指出根位置
-g	使用该选项可以用类文件生成调试表。当使用 jdb 时该选项可以提供完整的调试信息
-nowarn	关闭警告。当使用该选项时，编译器不再产生任何警告信息。注意这个选项只能在 1.1 和 1.1 以上版本中使用，在 JDK1.0 中不能使用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

选　项	描　述
-O	打开优化处理。将所有的 static、final 和 prive 方法设置成内联式的。虽然这样做可以提高程序执行的速度，但是它将导致类文件变大
-verbose	打开冗余编译。编译器打印正在编译和加载的源文件
-depend	使用该选项使得编译器考虑重新编译从别的类文件中引用的类文件。一般情况下再编译是根据文件日期进行的。注意：它适用于 1.2 版本的 JDK，不适用于 1.0 版本的 JDK
-Jjavaoption	该选项用于向实际运行该程序的 Java 解释器传递一个单独变量。Javaoption 不应当包含空格；如果需要包含空格，则应使用多重参数 -J。使用该选项允许用户传递像 mx 或 ms 这样的选项来改变编译器执行时的内存大小

4.5 javap，Java 反编译器

Java 反编译器用于分析已经编译过的 Java 字节码。

经过代码反编译以后，关于成员变量和方法的信息被打印出来。Java 反编译器的句法是：

java [选项] 类名

多重类也可以被反编译。使用单个空格分隔开每个类。反编译器可以使用的选项如表 4.3 所示。

表 4.3 javap 选项

选　项	描　述
-version	显示 JDK 的版本
-c	分解源文件并显示由编译器产生的字节码
-l	打印局部变量表
-public	只显示公共类和公共成员
-protected	显示保护的类和成员与公共的类和成员
-package	打印私有的、保护的和公共的成员变量和方法(缺省情况下 javap 使用该选项)
-private	显示所有的类和成员
-s	打印内部类型标志
-verbose	打印堆栈、局部变量和成员方法
-classpath dirs	在指定的 -DIRS. 目录中寻找类文件。对于多重目录，可以使用冒号或分号将它们分隔开。例如，在 DOS 机器上类路径可能被设置成 set CLASSPATH = .; C: \ users \ dac \ classes;c: \ tools \ java \ classes 这样
-verify	在源文件上运行验证器并检查类的调用

4.6 Javah C 头文件和承接模块文件的创建器

javah 工具用于当需要使用 C 语言扩充 Java 代码时创建 C 语言头文件和承接模块。

javah 工具的句法是：

javah [选项] 类名

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在这里 Java 类文件是没有 .class 扩展名的。表 4.4 中列出的是 javah 的选项。

表 4.4　javah 选项

选　项	描　述
- version	打印版本号
- help	打印帮助内容。与 javah 的 - help 选项一样
- jni	创建在 JNI 中使用的头文件
- td	识别 javah 使用的临时目录
- trace	跟踪加到承接模块文件中的信息
- classpath	为 javah 指定类路径
- stubs	创建承接模块文件来代替缺省的头文件
- d dir	指定创建头文件和承接模块文件的目录
- v	打印创建头文件和承接模块文件时的状态
- o filename	为存放承接模块文件和头文件的文件指定文件名。该文件可以是标准的文本文件或头文件(FILENAME.H)或承接模块(FILENAME.C)文件

4.7　javadoc 工具(文档生成器)

javadoc 工具用于在 Java 源文件中生成基于标志的 HTML 文件,这些标志以注释的形式嵌套在/ * * * /内。这些 HTML 文件用于存储关于类和方法的信息,这些类和信息很容易在网页浏览器中看到。

javadoc 实际是由 JDK 的创建器使用的,用于创建 Java API 文档(详细信息可参考 http://www.javasoft.com/doc)。用户可以在线查看 API,也可以查看存放在 \ JDK 1.2 \ SRC \ JAVA 目录中用于产生 API 的源代码。表 4.5 列出了与选项有关的信息。表 4.6 列出了与标志有关的信息。

表 4.5　javadoc 选项

选　项	描　述
- verbose	显示更多的与提供文档的文件有关的信息
- d directory	指定存储和生成 HTML 文件的目录。例如 javadoc - d c: \ users \ dac \ public _ html \ doc java.lang
- classpath dirs	在 DIRS 指定的目录里查找包括在源文件中的类文件。对于多个目录,可以使用冒号或分号将每个目录分隔开。例如,在 DOS 机器上 classpath 可能是下面这样: CLASSPATH = .;C: \ users \ dac \ classes;c: \ tools \ java \ classes
- sourcefile dirs	指定用冒号分隔的用于存放需要使用的文件列表的目录
- doctype	指定输出信息的文件类型。缺省值是 HTML,但也可以是 MIF
- nodepreciated	使 javadoc 忽略@depreciated 段落
- author	使 javadoc 利用@ author 段落
- noindex	使 javadoc 不创建索引文件
- notree	使 javadoc 不创建树文件
- j < flag >	指定直接传给 Java 运行器的标志

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 4.6 javadoc 标志

标 志	描 述
@see class	在 HTML 文件中增加一个 See Also 连接到由 class 指定的类上
@see class # method	在 HTML 文件中增加一个 See Also 连接，该连接指向由 method 指定的方法
@param param descr	描述方法参数
@version ver	指定程序的版本
@author name	在 HTML 文件中包含作者名
@return descr	描述方法的返回值
@exception class	创建一个指向由 class 指定的类的连接，由该类处理异常

4.8 jdb，Java 调试器

Java 调试器是 Java 环境的调试工具，它完全是命令行驱动的。用户可以使用该调试器调试本地系统上的文件，也可以调试远程系统上的文件。对于远程 Java 文件，jdb 必须使用 - host 和 - password 选项，这两个选项在表 4.7 中有所描述。jdb 由一系列命令组成，本章只介绍其中一部分。表 4.7 列出了与 jdb 选项有关的信息。

表 4.7 jdb 选项

选 项	描 述
- host hostname	告诉 jdb 远程 Java 程序的位置。hostname 是远程机器的名字(例如 well.com 或 sun.com)
- password password	对于远程 Java 文件向 jdb 传送口令，由 Java 解释器使用 - debug 选项发出

到目前为止所有的 JDK 工具都已经全部介绍过了，接下来我们介绍一下 CLASSPATH 变量。

4.9 CLASSPATH 环境变量

各种 JDK 工具只有一环境变量可用，即 CLASSPATH 变量，它是十分重要的变量，必须设置正确。如果设置有误，编译器、解释器和其它 JDK 工具都将无法找到 .class 文件，那么就无法完成各自的任务。

CLASSPATH 变量指向所有用于引入的类所在的目录。CLASSPATH 允许用户把自己的文件放在不同的目录中，但应使 JDK 工具知道它们所在的目录。

在 UNIX 机器上，CLASSPATH 变量是用冒号分隔一系列目录的。如下所示：

setenv CLASSPATH .:/users/java/:/usr/local/jdk1.2/classes/这个命令可以放在 .login 文件中，这样可以在每次登录时自动设置。

在 DOS 机器上，这个变量是用分号将一系列目录分隔开的，如下所示：

set CLASSPATH = .;c: \ users \ dac \ classes;c: \ tools \ jdk1.2 \ classes

这个命令行可以放置在 AUTOEXEC.BAT 文件中，这样可以在每次启动计算机时自动地设置 CLASSPATH 变量。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

命令中的第一个圆点指示的是 CLASSPATH 当前的工作目录，这个圆点非常有用，当不愿意输入完整的当前目录名时可以用它来代替，Java 程序允许这样做。

UNIX 版的 JDK 与 WIN32 版的 JDK 非常类似，并且大部分做同一工作的命令也很类似。但是 Macintosh 版本的 JDK 则有很明显的不同。

4.10　Macintosh 问题

因为 Macintosh 没有命令行界面，所以它的 JDK 工具与其它平台上的 JDK 工具有所不同。

注意：最值得注意的不同是 Macintosh 平台上的 JDK 工具不如其它平台上的 JDK 那么多。这种状况有希望得到改变，但是在这种改变到来之前，Macintosh 用户缺少某些最基本的工具，像 Java 调试器、javadoc 和 Java 反编译器。

Macintosh 版本的 JDK 包括四个工具：

- AppletViewer　在浏览器之外运行小应用程序的小应用程序浏览器。
- Java Compiler　把 Java 文件编译成 .class 字节码。
- Java Runner　Java 解释器，与前面介绍的“java”相似。
- JavaH　C 头文件生成器，能够创建承接模块文件，与“javah”相似。

对于大部分组件来说，它们做的工作与非图形界面相应工具做的工作是相同的，但界面是不同的。某些工具，像小应用程序浏览器与其它平台上的 JDK 版本中的小应用程序浏览器是很相像的。其它的工具如 Java Runner 则完全不同。接下来我们介绍这些与其它平台上相应组件有差别的工具。

4.10.1　Macintosh 平台上的小应用程序浏览器

打开的 Mac 小应用程序浏览器，它拥有标准的 Mac File 菜单和 Edit 菜单。另外还有一个状态框，状态框显示了当前分配给 Applet-Viewer 的 Java 虚拟机的内存空间和实际占用的内存空间。框中还显示有指示加载到小应用程序浏览器的信息加载进度条，如 .class 文件或 GIF 图像文件的加载。

注意：如果用户正在使用支持拖放的 Macintosh（7.1 及以上版本的 Mac OS），用户可以通过将包含有 <applet> 标志的 HTML 文件拖进小应用程序浏览器的小公爵图标来启动并运行小应用程序。用户也可以双击小应用程序浏览器的公爵图标并使用两个 open 菜单中的一个打开小应用程序。

小应用程序浏览器的 File 菜单包括下列选项：

- Open URL　打开某个包含小应用程序的网页的 URL。
- Open Local　打开标准的 Mac Open 对话框，并从中打开本地硬盘驱动器中的 HTML 文件。
- Save　不直接做任何事情，只是按照 MAC 的人性化界面的指导做。
- Close　如果顶层窗口可以关闭就关闭该窗口。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- Properties 显示小应用程序浏览器的安全特性。这些设置允许用户配置网络环境下的代理服务器和代理接口框。Network Access 框允许用户选择访问的方式——受限制的和不受限制的,不受限制的用户可以用小应用程序浏览器访问其它的类。
- Quit 关闭所有打开的小应用程序并退出小应用程序浏览器。

另外,小应用程序浏览器还有 Edit 菜单,但是它不允许写。在不久的将来有希望允许写,目前至少不必在 OPEN URL 对话框中输入长长的 URL。

当小应用程序正在运行时有一个 Applet 菜单出现。菜单中可用的命令如下所示:

- Restart 用当前的设置重新启动小应用程序。
- Reload 重新加载小应用程序。当类文件发生了改动时,需要重新加载。
- Clone 克隆当前的小应用程序,使用同样的设置新建一个小应用程序浏览器事例。
- Tag 显示 HTML 小应用程序标志,这个标志用于运行当前的小应用程序,另外 Tag 显示从 HTML 标志传给小应用程序的参数(见图 4.1)。
- Info 显示与小应用程序有关的特殊信息,这些信息是在小应用程序的程序中设置的(见图 4.2)。
- Properties 显示小应用程序浏览器的安全特性。这些设置允许用户配置网络环境的小应用程序浏览器;包括防火墙、HTTP 代理、使用有关的代理服务器和代理端口框。Network 框允许用户选择网络访问的方式。它的选项是 NO Network Access、Applet Host (缺省值)和 Unrestricted。Class Access 框允许用户选择访问的方式——Restricted 或 Unrestricted ,Unrestricted 允许小应用程序浏览器访问其它的类(见图 4.3)。
- Quit 关闭小应用程序浏览器窗口并结束小应用程序的运行。

4.10.2 Java Runner , Macintosh 平台上的 Java 解释器

MacJava Runner 是 Macintosh 平台上与前面描述的“ java”对等的解释器。因为 Macintosh 没有命令行,它是用基本的图形用户界面来设置各种选项的。为了使事情不至于变坏,该图形用户界面没有严格地遵循苹果公司的人格化的界面准则,这就意味着它的菜单与正常的菜单不同。

启动 Java Runner (Java 运行器)最普通的方法是双击一个拥有 main()方法的 .class 文件。用户使用 Java 编译器创建 .class 文件,它出现在桌面上或某个文件夹中,在文档的图标中央有公爵标志,在图标的左上角有多个 1 和 0。

另外用户也可以将 .class 文件拖到 Java 运行器图标上,或者双击 Java 运行器图标并在 Open File 对话框中出现的 .class 文件里选择一个。这两种方法都可以启动 java 运行器。

Java Runner 菜单是被作为 Apple 的子菜单隐藏起来的。所以 Java Runtime 菜单不会与正在运行的 Java 应用程序建立的菜单相冲突:

- Edit Mem 允许用户设置最大的和最小的堆空间,允许用户禁止异步无用单元收集(加快速度)。
- Edit Classpath 这个选项当前不能使用。
- Redirect Stderr 将错误信息重定向到用户在 Create File 对话框中指定的文件中,当使用了该选项后就会出现 Create File 对话框。
- Redirect Stdout 将程序信息重定向到用户在 Create File 对话框中指定的文件中,当使用了该选项后就会出现 Create File 对话框。
- Save Options 存储其它菜单设置。

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友, Java群空间; 可以联系群主获取更多大型企业内部技术教程。

- Save Text Window　将最顶层的窗口(例如 HelloWorld 程序的输出窗口)存储到文件中。
- Close Text Window　关闭顶层的文本窗口。
- Quit　退出 Java 运行器并结束所有应用程序的运行。

4.11　Java 编译器

Java 编译器有一个基本的图形用户界面,用户可以设置那些在其它系统上用作命令参数的选项。用户既可以通过拖动 .java 文件到编译器上来编译文件,也可以通过选择文件来编辑文件。其它的菜单选项如下所示:

- Close　当选中时好像是返回一个错误。这个选项在将来的版本中可能要修改。

警告:在 1.02 版本的 MacJDK 中,当出现方法找不到异常时这个菜单项才出现。在错误被纠正之前不要使用 Close 菜单项。

- Properties　打开一个对话框,允许用户通过复选框和其它项来设置选项以使其可在其它操作系统上使用。它也允许用户选择编辑器。缺省的编辑器是 simple text。该对话框还允许用户为编译器设置 CLASSPATH、.class 文件写入的目标文件夹和当有多个线程运行的情况下禁止其它线程来提高编译速度。
- Quit　退出编译器。

4.12　JavaH:C 头文件产生器

有了 JavaH 用户就可以把本地方法连接到 Java 代码中。这只适用于基于 PowerPC 的 Mac。在 Apple 菜单中除标准 Java 运行器之外没有它自己的菜单,连最重要的 Quit 命令都没有。要使用 JavaH 需要一个第三方编译器,比如 Metrowerks CodeWarrior,才能产生连接在 Java 代码里的 C 代码。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第二部分　Java 语言

第 5 章　面向对象编程
第 6 章　Hello world :用户的第一个 java 程序
第 7 章　数据类型及其它标记
第 8 章　方法
第 9 章　使用表达式
第 10 章　控制流
第 11 章　类
第 12 章　接口
第 13 章　线程
第 14 章　编写小应用程序
第 15 章　高级小应用程序代码
第 16 章　JAR 档案文件
第 17 章　小应用程序和应用程序的比较
第 18 章　管理应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 5 章　面向对象编程

本章内容：

5.1　面向对象编程：新的思维方式

5.2　简短的编程历史

5.3　对象

5.4　多实体对象

5.5　组织代码

5.6　对象和对象如何与 Java 类联系

5.7　建立层次：一种面向对象设计的方法

5.8　Java 并不是完全面向对象的语言

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

5.1 面向对象编程:新的思维方式

现在,作为程序员肯定听说过面向对象编程(OOP)这个奇妙的术语。OOP 是当今软件开发中最热门的一种方法。虽然它不是一个全新的概念,但是它需要很长一段时间才能走向大众化。Java 并没有像某些语言(如 SmallTalk)那样自称是面向对象的编程语言,但它确实是应用了其中的技术,用户完全可以得到这种技术的实惠。

面向对象编程是一个全新的编程方式。同时 OOP 的许多方面与结构化编程是相同的。当用户了解并掌握了与面向对象编程有关的新术语和新概念以后思想转变就十分容易了,到那时你可能不再注意你在做什么。用户可以看到 OOP 给程序员提供很多新的编写代码的方法,但却提供了许多新的程序设计思想。

在掌握了 OOP 编程的新思维方式以后,自然地要改变编写代码的词汇和语法的实际运用方法。遗憾的是掌握这种改变可能需要较长的时间。但从另外的角度来说每实现一点 OOP 编程都会为用户带来一份鼓舞。每个新的实现都会为用户打开一个新的编程可能性集。

5.2 简短的编程历史

要明白面向对象编程给程序员带来的好处,看一看编程技术发展的历史是有益的。

在早些时间的计算时代里,编程是一个十分花费精力的过程。计算机要做的每一步都必须详细入微(没有错误)地编写程序。早期的语言是著名的机器语言,后来进化为汇编语言。

如果你想指导别人系鞋带,这事情看起来简单,但教起来可不太容易,尤其你从未见过鞋带,那么你就很难做得到指导别人了。为了示范,你可以请同事(或者是一个不太讨厌这种要求的人)脱下他的鞋。让学系鞋带的人按照你说的方法去做。你可能会发现给出每一步精确的说明是十分重要的。"拿起鞋带的左端串到鞋带右端的下方,再拿起鞋的右端。"等等。

如果读者能够掌握教别人系鞋带的要领,那么就可以掌握编写类似的程序的方法。对于程序员来说指令的字面意思是含糊的(对于计算机来说是清楚的)给出指令说明是很重要的,比如"将寄存器 1 的内容压入堆栈","把累加器中的内容放在寄存器 1"等。

5.2.1 过程语言

程序员马上就可以看到更具风格的过程语言。这些过程语言把代码做成叫做过程或函数的块。每个块的目标是完成某个任务。例如,可以建立一个写屏的过程,像 Pascal 中的 writeln 或 C 的 printf 。这种程序的设计者认为用户总是在写不修改外部数据的函数。在 printf 或 writeln 例子中打印到屏幕上的串在打印前和打印后是同一个串。从根本上讲,这种思想不仅是建立一个黑箱上,而且在测试它时将黑箱彻底封闭。

使用这种方法的问题是必须用这种不修改它们绑定的外部数据的方式编写所有用到的函数。这样做起来是非常困难的。例如,是否传递在方法中"活"的用户希望更新的值?通常用这种方式强迫过程传递是非常困难的。所以当函数开始改变其作用范围之外的数据(在 C 语言中是通过传递指针实现的)时就会引发耦合问题以及其它的困难。因为函数正在改变作用范围外面的数据,所以测试工作就变得越来越困难了,耦合意味着每个方法必须不仅单独测试,并且要保证它改变的值仍然有效。另外还意味着每个黑箱必须与其它所有黑箱一起测试。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

如果这些黑箱有任何改动，它的父箱必须重新测试，因为其它箱的改变，父箱可能就不能工作了。（听起来很复杂，是吗？）

当开发大程序时耦合问题更加令人头疼。测试这些程序非常吃力，软件管理员要掉许多头发下来。如果他们有幸保住他们的头发的话，那么用户就很难使用他们的程序。

5.2.2　结构化程序设计

接下来的是结构化程序设计。结构化程序设计不是改变使用的语言，而是提供了一个新的处理方法。使用结构化程序设计，程序员在开始编写代码行之前先把程序周密地计划一下。当开始开发程序时，首先要设计一堆框图和流程图，以阐述各函数之间的互相作用关系和数据如何流过该程序。这些繁重的预编写工作在某些情况是很有效的，但在多数情况下效果是有限的。这种程序设计方法的缺点是大部分工作是花在文档整理和其它一些不必要的设计上。

另外，当程序员不得不在编写代码之前对他们所有的代码进行预设计时，程序设计就变成了语言翻译了。在程序实际开发之前要做许多试验。结构化程序设计要饱受这种开发循环的折磨。因为程序必须在各种功能被实现之前设计出来，所以程序员就没有多少自由，也不能独立地开发系统的单个部分。

5.2.3　面向对象编程

到了最后，终于进入了面向对象编程。面向对象编程要求很小的语言变化，但也改变了程序员的编程思维方式。他使用的编程技术吸收了过程语言的黑箱技术、接收了结构化程序开发的优点（实际上还做了进一步提高），更重要的是鼓励创造性的程序设计。

OOP 的对象是系统的，不仅仅是数据结构。对象也不仅仅是数、如整数和字符；他们还包含与数和数据结构有关的方法。在 OOP 编程里不仅仅有公开的系统中的数据传递，而且还有对象间的信息传递。不仅仅通过函数获取和传递数据，而且还向对象发送消息告诉对象执行它的任务。

面向对象编程并不是全新的；早在 1970 年 Xerox Parc 开发图形用户界面（GUI）、Ether-Net 和许多在今天看来很普通的其它的产品的研究组就开发了面向对象编程。为什么 OOP 经历了这么长的时间才走向大众呢？因为在开发中 OOP 要求规范地转移。另外，因为当时计算机处理能力有限，使用 OOP 方法开发的程序要求计算机拥有巨大的计算能力才能得到同样的运行效率，但在当时突破这一点的可能性是不大的。

对象本身是面向对象编程的奠基石。对象的概念可能是进行 OOP 设计必须理解的第一个也是最重要的概念。

对象是包括数据和方法的包。对象可在不破坏预定义码的情况下进行复制和调整。方法的用途是很容易定义的，一般情况下方法不预先设定太多潜在的用途，只要能处理眼下要处理的数据即可。当程序的需求有所扩充时，可以通过复制或调整方法来满足新系统的需要，因为面向对象的原因，用户不必更换方法。

5.3　对象

在用户工作过程中，可能对某些对象非常熟悉，比如计算器、电话、传真机和收音机，它们都是现实世界中对象的实例。当用户与这些对象打交道时，用户不会把对象与它的量和使用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

方法分开。例如，当用户打开收音机时，不关心什么是量（如电台的波段数）什么是方法（如调谐和音量控制），只是打开收音机，选定波段，然后坐下来听音乐。

当使用面向对象编程时，用户同样是如此处理的。作为结构化程序的程序员，用户肯定习惯于建立数据结构来存放数据并定义方法和函数来操作这些数据。而对象则把数据和代码结合在一起。这种合作的关系就是一个描述存在的事物与工作关系的对象。

现在看一下用户使用汽车的例子。当用户描述一辆汽车时会涉及几个重要的物理因素：汽车可乘载的人数、汽车运行的速度、汽车发动机的功率和制动系数等。另外，汽车还有几个功能定义：提速性能、减速性能、转弯性能和停放所需的空间。仅仅用物理因素或功能都不能单独定义用户的汽车，必须用它们两者一起来定义汽车。

5.3.1 传统的程序设计

在传统的程序中，用户可以定义一个如下所示的名为 MyCarData 的数据结构：

```
public class MyCarData {
    int weight ;
    float speed;
    int hp ;
    double dragCoef ;
    }
}
```

然后创建一套处理这些数据的方法：

```
public class RunCar
    public void speedUp (MyCarData m){
    ...
    }
    public void SlowDown (MyCarData m){
    ...
    }
    public void stop (MyCarData m) {
    ...
}
```

5.3.2 OOP 方法

在 OOP 编程中，汽车的运行方法和汽车的实际运行是结合在一个对象中的：

```
public class Car {
int weight ;
float speed ;
int hp ;
double drag coef ;

public void speedUp(){
    speed + = hp/weihgt ;
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public void slowDown (){
    speed  - = speed * dragcoef;
}

public void stop (){
    speed  = 0 ;
}
}
```

在每个新方法中，不再需要使用圆点引用变量(如 m.speed)，也不再需要向变量传递引用(如 MyCarData m)。同属于一个类的方法清楚地知道同属于一个类的变量。(这些变量也叫做域变量。)

5.3.3　通过继承扩充对象

现在我们基于一个超级对象来创建多个对象。回到汽车的例子上。Saturn SL2 是一辆汽车，并且肯定拥有一些不是所有汽车都有的属性。当生产汽车时，制造商并不是从零开始的。制造商知道他们的汽车需要几样东西：轮胎、车门、方向盘等。通常有许多部件在几种汽车中是通用的。那么就可以先装配通用部件，再装配 Saturn 的特有部件，然后在此基础上(因为每个 Saturn 汽车都有这些特殊的东西)再建造 SL2。这样做不是更好吗？

继承是 OOP 编程的特性，它允许用户只构造特殊的部分。通过把汽车共有的特性继承给 Saturn，也就是说没有必要每次都重新发明创造对象(汽车)。

另外，把汽车的特性继承给 Saturn 这种好处叫做多态性，Saturn 也是汽车。尽管看起来很容易，但是这种因素可延展的范围是巨大的。在传统的编程技术中，用户必须分别地处理每种汽车：这里的 FORD、那里的 GMC 等等。但在 OOP 中，所有汽车具有的特性被封装在 car 类里。当用户把 car 继承给 ford、GMC 和 Saturn，那么这些汽车就都具有了普通汽车具有的特性了，这样可以大大地减少工作量(不是完全地)。

例如说用户有一个追逐赛跑程序，在这个程序中有绿灯，黄灯和红灯。每个参赛车手都使用不同的汽车，每个赛手的汽车都有自己特殊的东西(比如 Volvo 有比较好的提速性能)。当把每个车放在赛道上，就等于给出了每个车的参考位置。赛道控制器不需要访问用于访问提速器或 CD 播放器的方法；对于每个车来说那些方法是私有的。当黄灯亮时坐在控制塔前的人需要能够告诉所有选手减速。因为处理的是汽车，所以控制塔程序把它们当做汽车来接收。看一下下面这段模拟的代码。

这里有两辆汽车：

```
class Lamborghini extends Car {
    public void SuperCharge(){
    for (int x = 0; x < infinity ; x + + )
    speedUp ();
    }
}
class Volvo extends Car {
CDplayer cd ;
Public void goFaster (){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    While (I-Have-Gas ){
        SpeedUp ();
    }
}
public viod jam(){
    cd.turnOn();
}
}
```

这里是追逐赛本身：

```
class RaceTrack {
    Car theCars [] = new Car [3];
    int numberOfCars = 0;

    public void addCar (Car newCar ){
        theCars [numberOfCars ] = newCar ;
        numberOfCars + + ;
    }

    public void yellowLight (){
        for (int x = 0; x < numberOfCars ; x + + )
            theCars [x].slowDown ();
    }
}
```

这里是放在一起的程序：

```
class RaceProgram {
Lamborghini me = new Lamborghini ();
Volvo       you = new Volvo ();
RaceTrack rc = new RaceTrack ();

public void start () {
rc .addCar (me )
rc .addCar (you );
while (true ){
if (somethingIsWrong )
rc .yellowLight ();
}
}
}
```

这个程序是如何工作的呢？在 RaceProgram 类中，用户建立了两个不同的对象：me（Lamborghini 类型的）和 you（Volvo 类型的）。如何将汽车以参数形式调用 rc.addCar？答案就是多态性。因为两辆汽车都是用 Car 扩展来的，并且又有各自的型号。这就意味着如果用户还想创建另一个类型的汽车（Saturn），用户只要调用 rc.addCar（the Saturn ）即可，不必对 RaceTrack 做任何修改。请注意即使 Volvo 拥有不同的结构也不必修改，因为它包含有 CD-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Player 变量。

5.4　多实体对象

作为过程语言的程序员可能出现的错误之一是把程序中的数据看作固定的量。屏幕就是一个很好的例子。通常过程化程序倾向于写点东西到屏幕上去。使用这种方法的问题是当用户切换到窗口环境并写多个屏时整个程序就危险了。这里只需要返回去对程序做一点点修改工作就可以把正确的数据写到适当的窗口上了。

相比之下 OOP 程序是把屏幕不是作为唯一的屏幕而是作为一个屏。增加窗口就像告诉函数写不同屏幕对象一样简单。

这只是展示了 OOP 的一个方面，在这里节省了许多编程的时间。当用户成为 OOP 程序员时，用户就开始考虑处理对象了。不管有一个对象还是一百个对象，都不会影响程序的编写。如果用户不熟悉 OOP 编程可能不会意识到这一点。总之用户对自己说的是："如果我有两个屏幕，当我准备打印某些东西到屏幕上时我需要确保打印在正确的屏幕上，并且会关注每个不同的窗口的用户交互。"

不论相信与否，在 OOP 环境下这种需求是不存在的。经过窗口和屏幕有效地抽象以后，当用户编写方法时，与要写的屏幕不相干了。窗口对象为用户处理这一切具体的操作。这实际上是多态性需要处理的事情，因为用户所有要关心的事情是：项目就是一个屏，而不是某个特殊的屏幕所具有的特殊能力。用户只要写这个项目就可以达到写屏幕的目的(译者注)。

5.5　组织代码

OOP 可以漂亮地组织代码是因为两个关键的因素：

- 当正确地使用 OOP 时，OOP 会强迫用户把代码组织成许多可管理的片段。
- 通过使用 OOP，每个片段都被组织成独立的片段，用不着用户关心组织。一方面 OOP 强迫用户组织代码，另一方面又组织独立。这是令人惊奇的特性。

5.6　对象和对象如何与 Java 类联系

Java 的核心就是对用户已经听说的对象的支持。它们以类的形式出现。(实际上 Java 有一个类名叫对象即 Object 类，所有的类都是从它那里继承来的，所以所有的类实际上都是 Object 类)。

对象是类的实例。在某种意义上可以认为类是创建对象的模板。拿一个矩形为例。用户要创建的矩形应该有 X 坐标和 Y 坐标、高度、宽度、移动方法和尺寸变化方法(收缩和放大矩形)。当为矩形编写代码时，用户可以在类里创建它。为了使用代码，用户需要建立该类的实例。这个实例就是一个简单的 Rectangle (矩形)对象。

5.7　建立层次：一种面向对象设计的方法

当第一次开始开发 OOP 程序时，有一个配方是很有帮助的，开发好的 OOP 结构很像烧制

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

一块馅饼。最重要的是要把所有配料收集齐全，然后再开始把它们包成一个馅饼。

5.7.1 把代码分成最小的实体

当编写一个 OOP 程序时，最重要的是确定出所有的组成部分(即所有的配料)。举例来说，如果用户正在编写一个狩猎游戏，确定出游戏中的每个东西是最重要的：动物、火药、人、子弹等等。

在组织完这些物体以后，用户需要把它分成更小的实体，包括数据和函数。如果用户正开始编写狩猎游戏，用户应该为四个项目创建一个实体表：

对象	实　　体
动物	位置、大小、威力等级、攻击能力、机动能力
火药	位置、威力等级
子弹	能够开火、大小和数量
人	从用户这里接收命令的能力，根据命令在迷宫中移动的能力、攻击能力、位置和大小

5.7.2 在实体间寻找共同点

开发 OOP 结构的下一步是在各个实体之间寻找共同点。例如，首先用户可能意识到动物和人的不同(不是指如何工作)是他们的控制者不同。动物是由计算机控制的，而人是由用户控制的。这样就可以把人和动物相同的代码用一套代码写出来，这部分的量较大；然后再为他们分别编写描述他们移动的代码，这部分相对较小；这样一来需要用户编写的代码是不是就可以大大地减少了。

无论用户感受如何，但都会认为真正实现起来并非易事，用这种方法处理对象正是该 OOP 示例的全部。

5.7.3 在实体之间寻找差异

下一步是在实体间寻找差异。例如子弹移动而火药静止。动物是由计算机控制的，而人是由用户控制的。在程序中用户要寻找出各个实体所特有的关系。

5.7.4 在实体间寻找最大的共同点

第三步是在程序中找出所有实体间最大的共同关系。在所有实体间很少能找出大家都具有的共同关系。但找出某个实体与其它实体的不同之处倒是比较容易。

继续看游戏示例，看所有四个对象有什么共同点？用户可能很快列出包括大小、机动能力(火药不需要移动)和位置的共同点。

他们之间是否有共同点存在？可能最明显的共同点不是前面所说的那条，而是都需要具有写屏的能力。这种能力非常明显，这也正是人们容易忽视的。不要忘记在明显的关系中寻找共同点。

使用这些实体用户可以建立一个名叫 Draw _ Object 类。这个类可以包括前面列出的各个项。

5.7.5 把剩下的共同对象放在一起并重复

下一步是将仍然有共同点的对象放在一起(在用户删除被编组到前面的类中之后)。用户使用这些共同点可以产生另一个层次的类，这些类的每一个都继承了包含所有公共信息(Draw _ Object)的类。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

回到游戏示例，在这一点上火药和子弹可以与人和动物分开。现在寻找出剩下的对象并重复这种寻找。

接下来用户发现火药和子弹的大小不同，运行速度也不同（火药的速度是 0）因为这些是细小的差异，用户可以把它们放在一个类中。

当用户查看动物和人时，用户会发现人包含的特性比动物多，所以就用 Main _ Character 类继承 Creature 类。

最后类的层次如图 5.1 所示。用户自己可以试着设计一个。这种图可以有不同的样式。

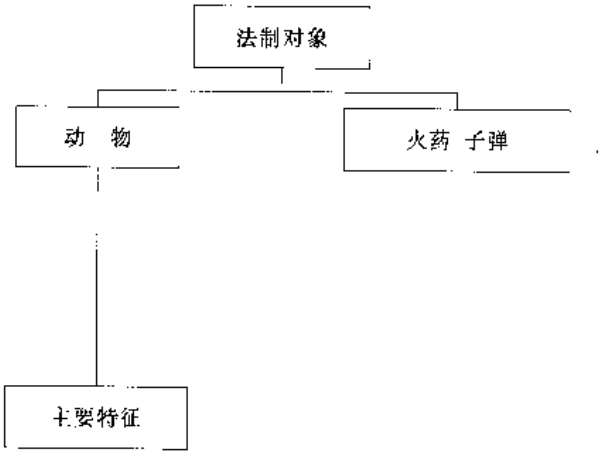


图 5.1　为游戏建立层次以便减少编写代码时间

5.7.6　利用对象扩充程序

当编写游戏时经常希望在狩猎游戏中增加许多攻击性的动物。因为 Creature 类是封装的，所有需要用户做的事情是创建动物和为它们确定位置并把动物加到列表。很明显用户不必再创建像 creature1Speed 、creature2Speed 、creature1Power 、creature2Power 等的变量。

用户可以考虑建立其它的变量。比如假设用户是其它语言的程序员，用户可能习惯于建立像整数变量这种具体的含意很狭隘的概念。但在 OOP 中用户可以直接建立一个动物，当然动物具有的内含要广泛得多。

5.8　Java 并不是完全面向对象的语言

本章的重点是介绍 OOP 概念。在本章中有意避免使用复杂的代码实现所述概念，在本书后面的部分将介绍许多代码。

用户已经看到了许多 OOP 编程的基础，下面可以真正地建立它们。Java 并不是完全面向对象的语言。

在使用 Java 编写 OOP 示例时，仍有可能把它用 Java 语言写成结构化程序。在不了解 OOP 工具如多态性和封装的情况下编写 Java 程序不是一个什么稀罕的事。

通过对 OOP 的介绍，有希望使用户放弃已经成为习惯的结构化编程的陋习。应当记住 OOP 与其它编程方法在思维方式上的不同之处。在本书中的有些小应用程序和应用程序是用两种方式编写的。观察那些分成多块的程序，然后当用户认为自己已经明白了 OOP 的时候，再来重读本章内容，看是否有所领悟。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 6 章　Hello World:用户的第一个 Java 程序

本章内容:

6.1　HelloWorld 应用程序
6.2　理解 HelloWorld
6.3　作为小应用程序的 HelloWorld 运行在 Netscape 中
6.4　关键字
6.5　API

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

6.1　HelloWorld 应用程序

当开始学习编写程序时,先编写一个短的应用程序,然后再研究语言的基础,这样做往往是很实用的。在本章中就是这样做的,到本章结束时读者就能够编写自己的 Java 程序了。

在第 3 章"安装并启动 JDK"里已经介绍了最简单的 Java 程序 HelloWorld。现在我们来仔细地看一下其中的每个组件。程序清单 6.1 显示了 HelloWorld 的源代码。

程序清单 6.1　HelloWorld 应用程序

```
public class HelloWorld {
  public static void main(String args[]){
    System.out.println("Hello World!!");
  }
}
```

就像用户看到的那样,这里确实没有太多的程序,这就是称它为最简单的 Java 程序的原因。不过用户还是应该仔细地看一下这个程序。在此之前先编译和运行一下这个程序。

6.1.1　创建文件

第一步:通过使用熟悉的文本编辑器(Windows 的 NotePad 或 Macintosh 的 simpleText)拷贝程序清单 6.1 到名为 HelloWorld.Java 的文件中来创建 HelloWorld 应用程序,调用 HelloWorld.java 很重要,因为编译器希望文件名与类标识符相匹配。(见本章后面声明类部分)。

警告:如果使用像 Microsoft Word 这样的编辑器输入代码,应确保以文本格式存储文件。如果用 Word 文档格式存储时,Word 在文件中增加了许多格式信息,这种文件不适合作 Java 文件的源代码它只会为用户带来麻烦。

6.1.2　编译代码

为了编译代码,用户需要首先安装 JDK。然后使用 JDK 的 javac 将程序清单 6.1 中的文本转化为计算机能够运行的格式。为了在 Macintosh 上运行 javac,可以将源文件拖到 javac 图标上,在其它计算机上则在命令提示符下输入 javac HelloWorld.java。

javac 程序用 HelloWorld.java 文件建立 HelloWorld.class 文件。HelloWorld.class 文件是字节码文件。字节码是可以运行在 java 解释器中的特殊指令。

参见第 3 章中"安装下载的 JDK"。

6.1.3　运行程序

既然用户已经编译了 HelloWorld 程序,那么就可以通过在命令提示符下执行下列命令来运行 HelloWorld 程序:java HelloWorld.

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:用在命令提示符下 java HelloWorld 中的 HelloWorld 既不是 HelloWorld.class 也不是 HelloWorld.java,而是该类的名字。

当做完这一工作后,计算机应该在屏幕上显示出下列内容:

Hello World!!

这东西看起来可能不太有趣,但它仅仅是一个简单的程序。如果在屏幕上看不到这个结果,请回去检查一下输入文件的内容是否严格地与程序清单 6.1 中的内容一致,并确保调用了 HelloWorld.java。

6.2 理解 HelloWorld

既然用户看到了 HelloWorld 程序的结果,那么就回到源代码处看一下是否能够理解它是如何工作的。就像用户开始看到的那样,HelloWorld 程序有许多部分。在完全理解它们之后,还需要很长的时间才能够编写程序。

6.2.1 声明类

创建任何 Java 程序的第一个任务是创建类。看一下 HelloWorld 应用程序的第一行:

public class HelloWorld {

这个语句声明了类的名字叫 HelloWorld。

参见第 11 章"类"。

为了创建类可以简单地写类似下面的一行字符即可:

public class className

在这里 className 是用户正在编写的程序的名字。另外 className 必须与文件名相呼应。给类起一个有含义的名字是一个好主意,这样做可以很容易地记住他们正在使用的东西。

注意:按照习惯类名总是以大写字符开头,这并不是要求的,但这是一个好的作风。在类的命名上还有许多限制,这在第 11 章"类"中将详细地介绍。

接下来注意左花括号"{",它位于类声明之前。如果用户看一下类的结尾,就会发现那里也有一个结束的右花括号}。这一对括号告诉编译器类从何处开始,到何处结束。在这两个括号之间的代码可以被认为是 HelloWorld 类,注意不要把括号搞混。括号一般用于把许多被称作块的东西括起来,在第 8 章"方法"中将更详细地介绍。括号是按照 LIFO(后进先出)规则匹配。这就意味着下一个结束花括号}与前面最近的一个开始花括号{是匹配的一对。在 HelloWorld 程序中有两个开始括号{,所以结束类定义的括号是最后一个。

6.2.2 Main 方法

接下来的 HelloWorld 程序行是:

public static void main (String arg[]){

这一行,声明的就是 main 方法。方法是最基本的小程序,每个方法都执行完整程序的某个任务。Main 方法执行的是应用程序最重要的任务,因为它位于所有 Java 应用程序的开始。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

例如当用户运行 Java HelloWorld 时 Java 解释器首先从第一行 main 方法开始解释。

当建立任何 Java 应用程序时,用户都需要创建 main 方法。在第 8 章中将详细介绍声明和使用方法。

6.2.3　写屏

当运行 HelloWorld 程序时"Hello world!!"文本是如何出现在屏幕上的呢? 答案(用户可能猜得到)就在于程序的下一行代码:

System.out.println("Hello World !!")

用户可以自己喜欢的其它字符替换引号("")里面的内容,当应用程序启动时 System.out 行得以运行,因为解释器找到代码的第一行并运行它,所以屏幕上会出现 Hello World!!。如果用户将其它的代码放在这里,解释器就运行其它的代码。

6.2.4　System.out 和 system.in

前面已经看到了 System.out.println 是用于在屏幕上显示文本的方法。事实上 System.out.println 可以被用作标准的文本输出。在大多数情况下,标准输出指的是屏幕.

System.out.println 的用途与 pascl 中 writln 的用途是一样的。在 C 语言中相应的函数是 printf,在 C++ 中相应的函数是 cout.

6.2.4.1　Println 与 print 的比较

下面的 println 用法有小小的不同:print("Hello World!!")。println 与 print 的不同在于 print 在打印完内容后不增加换行,这样后续的输出可以直接输出在同一行上。严格地讲 print 是 C/C++ 程序的 printf 和 cout 的远亲。

为了演示,可以将程序清单 6.2 的内容复制到名为 HelloWorld2.java 的文件中;HelloWorld2 扩充了 HelloWorld 示例。现在使用命令行 javac HelloWorld2.java 编译它。

程序清单 6.2　显两个打印输出的 Hello World 程序

```
public class HelloWorld2 {
  public static void main(String args[]){
    System.out.println("Hello World!");
    System.out.println("Hello World Again!");
  }
}
```

为了运行该程序,输入 java HelloWorld。这时用户应该看到下列输出结果:

Hello World!

Hello World Again!

注意这里的每句话都是占有单独的行。现在为了比较它们,我们试着用 print 来替换 println.将程序清单 6.3 中的内容复制到名为 HelloWorld3 的文件中,编译 HelloWorld3,然后运行它。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 6.3 使用 print 语句的 Hello World 输出

```
public class HelloWorld3 {
  public static void main (String args[]){
    System.out.print ("Hello World!");
    System.out.print ("Hello World Again!");
  }
}
```

用户应该注意到 HelloWorld3 的输出是：

Hello World ! Hello world Again !

是什么引起这一变化？当用户使用 print 时，程序不增加回车与换行。

6.2.4.2 扩充的串：在程序中写更多的内容

Java 从 C++ 中继承来的特性之一是把串加在一起的能力。虽然这一点看起来不合乎数字逻辑，但是对于程序员来说是极其方便的。重新访问最后一个 HelloWorld 程序并用 println 和 + 运算符可以得到同样的结果(见程序清单 6.4)。

程序清单 6.4 增加两个串的 Helloworld 输出

```
public class HelloWorld4 {
  public static void main (String Args[]){
    System.out.print ("Hello World!" + "Hello World Again!");
  }
}
```

当编译并运行 Helloworld4 时，用户应该看到与 Helloworld3 相同的输出结果。这可能看上去不那么有趣，那么现在看一下另外一种对字符追加的能力，用户还可以向字符串增加数字。例如，用户想在串中增加一个数 43。程序清单 6.5 显示了实现这一想法的程序。

程序清单 6.5 带数字输出的 Hello world

```
public class HelloWorld5 {
  public static void main (String args[]){
    System.out.print ("Hello World! " + 43);
  }
```

程序清单 6.5 产生下列输出结果：

Hello World! 43

6.2.4.3 使用 System.in 获取用户输入的信息

与 System.out 相对的方法是 System.in 。System.out 是用于向屏幕输出信息的，而 Sys-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

tem.in 程序是用来获取信息的。

6.2.4.4　请求用户输入

使用 System.in.read ()可以获取用户输入的字符。这个方法用得不多。因为 Java 程序不怎么使用 System.in;首先,因为它不适用于小应用程序(将在"作为小应用程序的 HelloWorld"部分介绍)。程序清单 6.6 显示了读取用户输入字符的 Java 应用程序例子。

程序清单 6.6　ReadHellow :读取用户输入的小应用程序

```
import java.io.*;

public class ReadHello {
public static void main (String args[]){
  int inChar;
  System.out.println("Enter a Character:");
  try {
    inChar = System.in.read();
    System.out.println("You entered " + inChar);
    } catch (IOException e){
      System.out.println("Error reading from user");
      }
  }
}
```

用户可能已经注意到这里又增加了许多代码。在解释这些新增加的代码之前,用户应该先编译运行一下,看是否能够得到下列结果:

Enter a Character:

A

You entered 65

首先,用户最感兴趣的代码是:

inChar = System.in.read( );

System.in.read( )是监视用户输入字符的方法。该方法返回一个值,这个值可以用在表达式中。在 ReadHello 中,变量 inChar 用于接收 System.in.read( )方法的返回值:

像程序清单 6.5 一样,在接下来的一行中将变量 inChar 的值加到 Systemout 串中。用户可以看到增加这个变量到串中的结果。其实用户不必使用变量,如果用户喜欢,用户可以直接在第二个 System.out 行中,如下所示:

System.out.println("You entered" + System.in.read( ));

注意程序在接收用户输入的字符后显示了一个数。这是因为 System.in 的 read( )方法返回的是一个整数,不是一个实际的字符。该数字是该字符在 ASCⅡ字符集中的值。

将整数转换为字符　用户需要把 System.in.read( )返回的数强制转换成字符(强制转换可以把一种数据类型转换成另一种)。修改后的 ReadHello 如程序清单 6.7 所示:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 6.7 ReadHello:从控制台读入字符的应用程序

```
import java.io.*;
public class ReadHello {
  public static void main (String args[]){
    char inChar;
    System.out.println("Enter a Character:");
    try {
      inChar = (char) System.in.read();
      System.out.println("You entered " + inChar);
      } catch (IOException e){
        System.out.println("Error reading from user");
        }
  }
```

注意 System.in.read( )前面的字符。(char)引起了整数向字符的转换。

其它的代码—try 和 catch 剩下的代码是干什么用的?在该程序代码中有一个被称作 try—catch 块的序列,它有着特殊的用途。

在某些程序语言中,当在执行过程中发生问题时,程序员无法捕捉和处理这些问题。当某些问题发生时,可能会导致系统挂起并结束程序的(通常伴有严重的消息如 General Protection Fault,或 Core Dump)运行。在某些任务中它比较复杂。在 Java 中大多数问题都会引发异常,异常可以被用户处置,所以程序不会停止工作。

参见第 20 章的"Java 的异常"部分。

参见第 20 章的"Java 的事件"部分。

6.3 作为小应用程序的 HelloWorld 运行在 Netscape 中

如果用户读完过本书肯定会有很大兴趣去用 Java 编写小应用程序。小应用程序可在浏览器里运行,比如 Netscape Navigator。

小应用程序和应用程序有几点不同。最重要的是 Java 小应用程序类扩展了已有的 Java 类。这个类叫做 Java.applet.Applet。到目前为止可以说:用户必须把 Applet 扩展成为能够这样使用的类。在本章后面将详细介绍这方面的内容。

参见第 34 章"Java 小程序与应用程序"。

6.3.1 新的源代码和编译新的源代码

最简单的小应用程序是 HelloWorld 小应用程序,HelloWorld 小应用程序的源代码如程序清单 6.8 所示。正如下面用户将要看到的那样,小应用程序 HelloWorld 与程序清单 6.1 中的 HellWorld 应用程序有着明显的不同。用户在本章后面将分解它的源代码并逐行理解它。现在将程序清单 6.8 中的代码拷贝到名为 HelloApplet.java 的文件中并编译该文件。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

程序清单 6.8　作为小应用程序的 Hellworld

```
import java.applet.Applet;
import java.awt.Graphics;
public class HelloApplet extends Applet {
  public void paint (Graphics g) {
    g.drawString ("Hello World!",0,50);
  }
}
```

6.3.2　创建 HTML 文件

在用户创建程序清单 6.1 到 6.5 中的 HellWorld 应用程序时,用户是使用 Java 解释器运行的。小应用程序不能用命令行来运行,它们必须在浏览器里运行。但是用户如何告诉浏览器打开小应用程序呢?

如果用户编写过网页,用户一定熟悉 HTML 页。HTML 页是用浏览器如 Netscape Navigator 处理的。为了把小应用程序装入浏览器,用户需要把 HTML 标志嵌入到 HTML 文件中。这样一来 HTML 文件就可以被读进浏览器中了。

最简单的使用 HellWorld 类的 HTML 文件如程序清单 6.9 所示。

将这些文本拷贝到名为 HelloApplet.html 文件中。

参见第 14 章的"在 HTML 页中包含 Java 小应用程序"部分。

看一下程序清单 6.9 中的第二行。注意到〈APPLET〉标志了吗?〈APPLET〉标志是一个新的用户包括 Java 小应用程序的 HTML 标志。在创建 HTML 文件时,不要忘记包括〈APPLET〉标志,否则小应用程序不会出现。

> **注意**:使用 Java 文件时使文件名和类名相同是非常必要的。这一点对于 HTML 文件却不是必要的。事实上一个 HTML 文件可以包含几个〈APPLET〉标志。

程序清单 6.9　HelloApp.html—使用小应用程序的 HTML 文件

```
<HTML>
<BODY>
<APPLET CODE="HelloApplet.class" WIDTH = 200 HEIGHT=200> </APPLET>
</BODY>
</HTML>
```

6.3.3　在小应用程序浏览器中运行程序

为了运行小应用程序,JDK 包括了一个非常简单的浏览器叫做小应用程序浏览器(AppletViewer)。小应用程序浏览器在 HTML 文件中查找 <APPLET> 标志并为每个小应用程序打开一个新窗口。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

当用户在小应用程序浏览器中运行小应用程序浏览器时，用户可以看到类似于图 6.1 所示的输出。为了使用小应用程序浏览器运行 HelloApplet 程序，可以在命令行里输入 appletviewer HelloApplet.html。

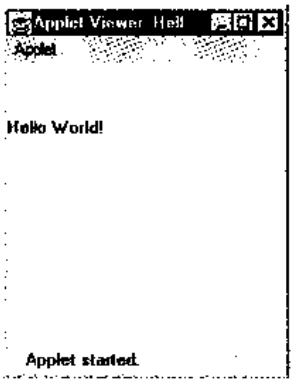


图 6.1 打开新窗口并运行 HelloApplet 的小程序浏览器

6.3.4 在 Netscape Navigator 中运行 HelloWold

运行小应用程序的另一种方法是使用 Netscape Communicator。用户可能已经熟悉了 Navigator 的使用。为了在 Netscape 中打开 HelloApplet 程序，请选择 File 并打开 File 菜单，然后选择 Hello Applet.html 文件，如图 6.2 所示。

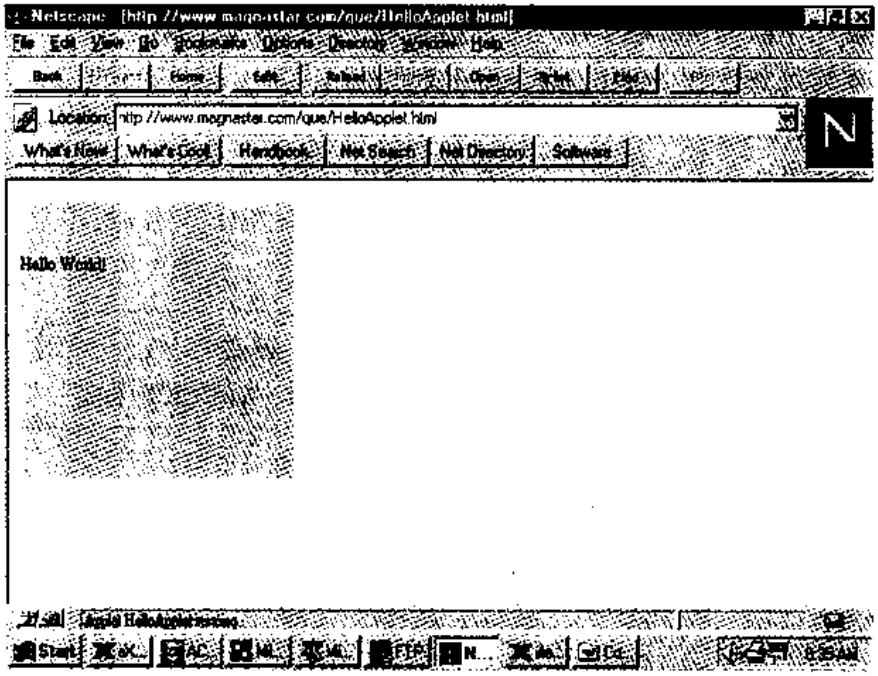


图 6.2 HelloApplet 也可以使用 Navigator 来运行

6.3.5 理解源代码

既然已经看完了如何运行 HelloApplet 程序，现在回过头来看一看程序是如何工作的。

6.3.5.1 引入其它的类

首先注意到的应该是代码的头两行：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.applet.Applet;
import java.awt.Graphics;
```

import 语句是新的语句。使用它是必要的,通过它可以很容易地使用早已建好的类的内容,用不着再在自己的程序中重新编写它。import 语句允许用户如此地使用其它类。如果用户熟悉 C/C++ 的 # include 声明,import 语句的功能在某种程度上与 # include 是一致的。

在 HelloApplet 程序中使用了两个类,这两个类是在 HelloApplet 之外的。第一个是 java.applet.Applet 类。Applet 类包括了所有小应用程序特有的信息。事实上,任何作为小应用程序运行在浏览器中的类必须扩展 java.applet.Applet。第二个引入 HelloApplet 的类是 java.applet.Graphics 类。java.applet.Graphics 包含各种屏幕绘图工具。事实上屏幕是被当作 Graphics 对象来对待的。

6.3.5.2　声明小应用程序类

用户可能注意到 HelloApplet 类的声明与 HelloWorld 应用程序中的 HelloWorld 类的声明有点不同。HelloWorld 扩展了 Applet。是否还记得在上一章中是如何建立类结构的? extends 是关键字,它指的是某个类应当进入那个层次。事实上扩展另一个类的类被放在已有链的前面:

```
Public class HelloApplet extends Applet{
```

参见第 11 章的"超类——扩展另一个类"。

用户可能认为这是个问题,但很重要的一点是:所有小应用程序必须使用完整的 java.applet.Applet 格式。因为用户引入了 Applet 类,所以用户可以直接调用 Applet。如果用户没有调用 java.applet.Applet,那么用户就必须仍然使用全名来扩展:

```
Public class HelloApplet extends java.Applet{
```

6.3.5.3　小应用程序方法—paint

另一个值得注意的 HelloApplet 类与 HelloWorld 类的差别是 HelloApplet 没有 main 方法。另外这个小应用程序却有一个名为 paint 的方法。这是怎么回事?

答案是:应用程序不是自启动的。小应用程序是被已经运行的程序(浏览器)加载的。浏览器预定义了每个小应用程序要做的东西。它是通过调用小应用程序具有的方法来实现加载的。这个方法就是 paint:

```
Public Void Paint(Graphics g){
```

当浏览器需要在屏幕上显示小应用程序时就调用 Paint 方法,用户可以使用 Paint 方法显示任何内容。浏览器是通过向 Paint 方法传递 Graphics 对象来帮助实现显示的。Graphics 对象给出 Paint 方法向屏幕上显示内容的具体做法。

下面是一个使用 Graphics 向屏幕书写文本的例子:

```
g.drawString ("HelloWorld!",0,50);
    }
}
```

6.3.6　小应用程序简短的生命周期

Paint 方法不仅仅是浏览器调用小应用程序的方法。用户可在 HelloWorld 例子中加载其

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

它的任何方法，就像加载 Paint 方法一样。

当小应用程序被加载时，浏览器调用 init()方法。不管用户返回同一个网页多少次，init()方法只调用一次。

在调用 init()方法之后，浏览器首先调用 Paint()方法。这意味着在使用 Paint()方法之前需要初始化某些数据，用户应该在 init()方法中完成初始化。

接下来调用的是 Start()方法。Start()方法在每次小应用程序页被访问时调用。这就意味着当用户离开某个网页后又按向后(Back)按钮时，Start()方法就再一次被调用。无论如何 init()方法不是这样。

当用户离开某个网页时(比如单击了某个超链)，stop()方法就会被调用。

最后，当浏览器全部退出时调用 destroy()方法。

> **注意**：与 Paint(Graphics g)方法不同，init()、start()、stop()和 destroy()方法的括号内部不带任何参数。

6.4 关键字

在开始进一步研究本章介绍的几个主题之前先让我们介绍几个需要了解的东西。

最重要的一个就是在 Java 中关键字的使用。所谓的关键字就是一定的字符序列，这些字符序列在 Java 中有特殊的含义。某些像动词，某些像形容词、某些像代词。有些关键字是为语言以后的版本保留的标志、还有一个(指 goto)是来自于古老的过程语言的遭诅咒的关键字，在文明的 Java 语言中可能根本就不用。

下面列出了 56 个可以在 Java 中使用的关键字。当用户弄懂所有这些术语的含义时，用户就已经在成为 Java 程序员的路上走出很远了。

abstract	**boolean**	**break**	**byte**
case	cast	catch	char
class	const	continue	default
do	double	else	extends
final	finally	float	for
future	generic	goto	if
implements	import	inner	instanceof
int	interface	long	native
new	null	operator	outer
package	private	protected	public
rest	return	short	static
super	switch	synchronized	this
throw	throws	transient	try
var	void	volatile	while

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

关键字 cast、const、future、generic、goto、inner、operator、outer、rest 和 var 是保留字,在目前的 Java 中没有含义。有其它语言如 C 、C++ 、Pascal 或 SQL 编程经验的程序员可能知道这些术语最终可能会被用到。眼下用户用不着关心这些术语,不使用这些术语的 Java 更简单并且更容易维护。

在这里没有列出 true 和 false;从技术上讲它们是 Boolean 变量或 Boolean 常量的字符值。

用户关心关键字的原因是因为这些关键字在 Paint 中拥有特殊的含义;用户不能把它们当作别的标识符。这意味着用户不能用它们作为名字来创建类。如果 HelloApplet 在上面这个列表中,那么编译器绝不会为用户编译它。另外也不能把它们当作变量、常量等来使用。但是它们可以被用作更长的标志中的一部分,例如:

Public int abstract _ int;

> **注意**:因为 Java 是大小写敏感的,所以用户可以通过把首字符大写来将这些关键字做别的标识符来使用。虽然这是可行的,但这是一个不好的主意,这样做容易使得程序的易读性变差,当代码需要升级时可能要浪费时间来重新理解源代码的含义:
>
> public short Long;
>
> 这样做是可以的,为了清楚、明了、易懂起见请不要这样做。

另外有许多类是定义在包中的。虽然包的名字不是关键字,但是乱用这些名字可能使得意义含糊,当别人将来使用这些应用程序或小应用程序时会增加许多误会。

6.5　API

在本章中用户了解了几个类的用法,并且编写了一个类。在这些类中最重要的类是 java.applet.Applet 类

为什么要告诉用户 java.aplet.Applet 的方法呢? 答案是 Java API 中的所有类都是有很好的文档注释的。虽然直到用户读完后面的几章后也不可能完全明白 API,但是现在就开始查看它们也是很重要的。

当用户逐渐成为程序员时,API 可能要成为用户最好的朋友。事实上 Java 丰富的 API 是它如此成功的一个原因。

用户可以在 Sun 公司的站点 http://www.javasoft.com/products/JDK/CurrentRelease/上访问 API 文档的超链版本。

当开发 API 时,用户应该注意各种类是如何使用 extends 关键字和如何从其它的类继承的。Sun 公司做了大量的工作努力使用户尽可能少地编写代码;用户必须学习使用这些类。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 7 章　数据类型及其它标记

本章内容：

7.1　Java 的两种数据类型
7.2　了解布尔变量
7.3　各种整数类型
7.4　运算符
7.5　字符变量
7.6　浮点变量
7.7　数组
7.8　空白
7.9　注释
7.10　源程序中的符号：赋值
7.11　创建和撤销对象

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

7.1　Java 的两种数据类型

使用计算机做工作，简单时就像学校里写作业，复杂时又像是在解量子理论里的方程式，但不论怎样，计算机做的工作都是处理数据。对于计算机来说数据可以是数、字符或简单的值。Java 使用了几种数据类型，在本章中将介绍其中最重要的部分。

在 Java 中数据类型被分为两类。

- 基本类型
- 引用类型

Java 有八种基本类型，每种类型都有各自的用途和用法。

- Boolean
- char
- float
- byte
- int
- double
- short
- long

在本章中将详细地介绍这八种数据类型。现在让我们先来看一下表 7.1，它显示了这八种基本数据类型的取值范围。

表 7.1　Java 语言的基本数据类型

类型	描　　述
boolean	只有两个值，true 和 false
byte	8 位带符号整数，可以表示 -2^7 到 2^7-1（-128 到 127）之内的任意数
short	16 位带符号整数，取值范围为 -2^{15} ~ $2^{15}-1$（-32768 到 32767）
char	16 位无符号整数，取值范围为 065535。作为字母数字它与高位置 0 的 ASCII 字符一样
int	32 位有符号整数，值域为 -2^{31} ~ $2^{31}-1$（-2,147,483,648 到 2,147,483,647）
long	64 位有符号整数，值域为 -2^{63} ~ $2^{63}-1$（-9,223,372,036,854,775,808 到 9,223,372,036,854,807）
float	使用 IEEE754 - 1985 标准（范围大约在 -10^{39} 与 10^{33} 之间）的 32 位单精度浮点数
double	使用 IEE754 - 1985 标准的 64 位双精度浮点数，取值范围大约在 -10^{317} 到 10^{317} 之间

与其它语言不同，Java 的基本数据类型是唯一的，表 7.1 显示的取值范围是固定的，不管用户使用的是什么类型的计算机，它都具有表 7.1 所示的值域。这一点也进一步增强了程序的安全性和可移植性，这一点其它的语言可能做不到。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

7.2 了解布尔变量

Java 中最简单的变量类型是布尔型变量。布尔变量有两个可能的值 – true 和 false。在某些语言中,布尔变量的值是 0 和 1;在 C(++)中 false 是 0,而其他数是 true。Java 将其简化成一位,并确定为 true 和 false。

在跟踪对象状态时经常用到布尔变量。例如,当说明一张纸是否在桌子上时,他的代码可以是:

```
boolean on _ the _ table = true;
```

7.2.1 声明变量

在进一步学习之前,用户应该明白前一部分最后的代码行的含义。当用户在 Java 中创建变量时,用户必须知道几件事情:

- 用户必须知道要使用的数据类型。在前一个例子中变量是 boolean 型的。
- 用户必须知道什么东西要调用变量(on-the -table),
- 用户也可能要知道变量的初值。在前一个例子中纸本来就在桌子上,所以把变量设置成 true。如果用户没有指定布尔变量的值,Java 编译器自动地把布尔变量的值设为 false。

用户可以用与前面显示的例子相同的方法设置任意其它的变量:

1.说明数据将使用的类型(boolean)。

2.指示变量名(on-the-table)。

3.为变量赋值(= true)

4.像其它的 java 代码行一样,使用分号结束行内容。

7.2.2 标识符:变量命名

还是看前面的那个例句:

```
boolean on _ the _ table = true ;
```

计算机如何处理构成 on _ the _ table 的这些字符呢? on _ the _ table 在编程词法中叫做标识符。标识符非常重要,因为它用于代表一个完整的东西。事实上用户可以选择由任意一组字符作为标识符来代表任何变量,如常数、类对象、标签、或方法。标识符一经建立,在同一个代码块中它就代表着相同的对象(如常数、类、对象,标签或方法等)。

下面是用户在建立标识符时必须遵守的规则:

- 第一个字节必须是字符。而后面跟着的可以是字符也可以是数字。
- 这些开头字符不能是拉丁数字或普通数字,它们可以是字符表中的任意字符。因为 Java 是基于统一码的,所以标准的标识符可以用在各种语言中,比如 Arabic-Indic、Devanagari、Bengali、Tamil、Thai 等等、等等。
- 下划线(_)和货币符号($)可以用在标识符中的任何位置,甚至用在开头。
- 像在 C 和许多其它的现代语言中一样,在 Java 中标识符是区分字符大小写的。这就意味着 on _ the _ table 与 On _ The _ Table 是不同的标识符。大写的标识符与小写的标识符所代表的变量是不同的。
- 要确保标识符能够表达完整的含义,不要怕长。短的标识符容易输入、输入得快捷、不

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

易出现错误，而输入长的标识符则更具有描述能力和可读性，用户在命名标识符时应兼顾这两方面的因素。在大的应用程序中选一种命名约定是很有用处的，这样可以减少相像和避免标识符重用（即一个标识符代表两个对象）。下面的作法是不提倡的，创造几个名为 X、X1、X2 和 X4 的标识符，用户很难记住每个变量的用途。另外标识符不能是关键字。

● 表 7.2 显示了几个合法的标识符和几个非法的标识符。第一个非法的标识符是因为它的数字开头。第二个非法的标识符是因为它包含有非法字符。第三个非法标识符也使用了非法字符：空格。第四个非法标识符使用的是数字，数字不能用作标识符。最后一个非法标识符包含有另一个非法字符：连字符，或叫做减号。Java 将最后一个非法标准符是当做表达式来处理的，它认为这是一个由两个量执行减法运算的表达式，其中含有两个标识符和一个运算符。

表 7.2　合法标识符和非法标识符的示例

合法标识符	非法标识符
HelloWord	9HelloWorld
counter	count & add
Hotjava &	Hot java
ioe _ queue3	65536
Eruest lawrence thayers	non - plussed FamousPoemOf June 1888

7.2.3　修改布尔变量

在第 10 章“控制流”中用户将会看到布尔变量可以用于改变程序的行为。例如，如果纸在桌上，用户不必做任何事情，但是如果掉在地上了，用户就可能告诉计算机把它捡起来。

有两种办法可以修改布尔变量，因为布尔变量不能用数字赋值，所以用户必须将布尔变量设成 true 或 false，第一种办法是显式地赋值。例如，如果用户有一个变量叫做 My _ First _ Boolean，为了将这个变量设成 false 用户应当输入下列内容：

```
My _ First _ Boolean = false ;
```

如果把这行声明与前面的 on _ the _ table 声明比较；用户可能会注意到它们是非常相似的。

第二种为布尔变量赋值的办法是基于等式或其它变量实现的。例如，如果用户希望 My _ First _ Boolean 拥有与 on _ the _ table 相同的值，用户应该输入下列内容：

```
My _ First _ Boolean = on _ the _ table ;
```

用户也可以用基于其它数的等式来设置布尔变量。例如，下面的语句将 My _ First _ Boolean 设置成 false；

```
My _ First _ Boolean = 6 > 7 ;
```

因为 6 是不大于 7 的，所以等式的左边的结果是 false。在第 10 章将详细介绍这种等式。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:布尔类型在 Java 中是新特性,与 C 和C++ 中的布尔类型的用法不同。这种严格的坚持看上去似乎有些沉闷。但另一方面却可以杜绝不良的动机,这种动机将会导致查找程序错误困难,从而浪费许多智力劳动。

7.3 各种整数类型

接下来要介绍的另一个 Java 基本类型是整数类型:

- byte
- int
- char
- short
- long

正如在表 7.1 中用户看到的那样,这几种类型的值域各不相同。例如,byte 类型不能表示大于 127 的值,但是 long 类型却可以很容易地表达国家级的信贷数目,它的值域比 byte 类型的值域大上百万倍。

应根据不同的情况使用不同的整数类型,用户不要因为 long 是最大的整数类型而到处使用它。一般的程序里没有太多长整型数据要处理,许多变量使用 short 就足以表示了。很重要一点就是 long 型整数在计算机内存中占用的空间远比 short 大。

7.3.1 整数类型值的限制

整数类型的取值范围如表 7.3 所示。

表 7.3 整数类型和它们的取值范围

整数类型	最小值	缺省值	最大值
byte	-128	(byte)0	127
short	-327868	(short)0	32,767
int	-3147,483,648	0	2,147,483,647
long	-9,223,372,036,854,775,808	0	9,223,372,036,854,775,807
char	0	0	65535

注意:最大的 long 数值比全世界所有人口 50 年内度过的秒数总和还要多。与一立方英里沙子中的沙粒数量差不多。即使是海量存储设备里存储的数据也达不到这么大。

注意:如果某个创建的数大于上面所示的范围时,系统并不产生溢出或异常。而是用它的补码值做为运算的结果(以 byte 类型为例,127 + 1 = -128、127 + 9 = -120、127 + 127 = -2)。但如果出现除数为零或取模运算的模数为零时 Java 则调用 Arithmetic Exception 来处理异常。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

7.3.2　创建整数变量

所有的四种主要的整数类型都可以使用几乎相同的办法创建(将在本章后面介绍 char 类型)。下面是创建这四种类型变量的示例:

```
byte    my-first-byte = 10
short   my-first-short = 15;
int     my-first-int = 20
long    my-first-long  = 25;
```

注意:整数类型的声明与布尔型变量的声明几乎一样,而几种整数类型的声明完全一样。与布尔型变量最主要的差别是整形变量必须赋以数字,而不是 true 或 false。另外应该注意整形变量必须用整数来赋值,不能使用小数。换句话说,如果想赋的值是 5.5 或 5and 2/3(五又三分之二),那么用户不能使用整数类型。在本章后面的"浮点变量"部分将详细介绍这种数字类型。

7.3.3　整数运算

用户可以对整数执行多种运算。表 7.4 完整地显示了整数的各种运算。

表 7.4　整数表达式运算

运　　算	描　　述
= 、+ = 、- = 、/ = 、* =	赋值运算符
= = 、! =	等于和不等于运算符
< 、< = 、> 、> =	不等于运算符
+ 、-	单符号运算符
+ 、- 、* 、/ 、%	加、减、乘、除和取模运算符
+ + 、- -	增量和减量运算符
< < 、> > 、> > >	移位运算符
< < = 、> > = 、> > > =	移位并赋值运算符
~	按位逻辑取反运算符
& 、! 、^	按位与、或、异或(XOR)运算符
& = 、! = 、^ =	按位与、或、异或并赋值运算符

在后面第 10 章中将详细介绍产生布尔结果的等于和不等于运算符。现在着重介绍算术运算符。

7.4　运算符

运算符用于改变某个特殊对象的值。例如,假如用户想让 10 加 5 或减 5。用户很快会看到,用户将使用加或减运算符。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

现在把表 7.4 中的运算符分成几类来介绍，C 程序员和C++ 程序员应该对表 7.4 中的运算符十分熟悉。

7.4.1 算术运算符

算术运算符用于对变量执行标准的数学运算。这类运算符包括：

+ 加运算符
- 减运算符
* 乘运算符
/ 除运算符
% 取模运算(给出除运算的余数)

在这些运算中用户可能只对取模运算不太熟悉。取模运算就是求出被模数除所剩的余数。换句说，10%的余数为 0，因为 10 能够被 5 整除。而 11%5 的结果是 1，因为 11 被 5 除的结果是得 2 余 1。

程序清单 7.1 显示了使用这些运算符的示例。

程序清单 7.1 使用算术运算符的例子

```
int j = 60;                    // 将变量 j 的值设置为 60
int k = 24;
int l = 30;
int m = 12L;
int result = 0L;

result = j + k;                // 结果为 84: (60 加 24)
result = result / m;           // 结果为 7:(84 除 12)
result = j - (2 * k + result);  // 结果为 5: (60 减去 48 和 7 的和))
result = k % result;           // 结果为 4:(24 除以 5 的余数)
```

7.4.2 赋值运算符

最简单的赋值运算符就是标准的赋值运算符。这个运算符也叫做取(gets)运算符，因为是左边的对象取右边的值：

= 赋值运算符

算术赋值运算符为赋值提供了一个捷径。当变量的现在值是决定赋值后值的一个因素时使用算术赋值运算符是很有效的：

+ = 加并赋值运算符
- = 减并赋值运算符
* = 乘并赋值运算符
/ = 除并赋值运算符
% = 取模并赋值运算符

除赋值运算符以外，算术赋值运算符好像是将运算符左边的变量放在右边来执行相应的运算，下面两行是完全等价的语句。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
X = X + 5;
X + = 5;
```

程序清单 7.2 显示的是使用算术运算符的示例。

程序清单 7.2　使用算术运算符的例子

```
byte j = 60;          // 将变量 j 的设置为 60
short k = 24;
int l = 30;
long m = 12L;
long result = 0L;

result + = j;         // 结果为 60: (0 加 60)
result + = k;         // 结果果为 84: (60 加 24)
result / = m;         // 结果为 7: (84 除以 12)
result - = l;         // 结果为 - 23: (7 减 30))
result = - result;    // 结果为 23: ( - ( - 23))
result % = m;         // 结果为 11: (23 除以 12 的余数)
```

7.4.3　增量/减量运算符

增量运算符和减量运算符只使用一个变量(也叫做一元运算符):

++　增量运算符

- -　减量运算符

例如,增量运算符为操作数加 1,如下行代码所示:

```
X ++ ;
```

它与下行代码是等价的:

```
X + = 1;
```

增量运算符和减量运算符放在变量的左边和放在变量的右边的作用有些不同。如果变量在运算符的后面(如 ++ X),那么就是先加后赋值。下面的程序段中 Y 的结果是 6:

```
int    X = 5 ;
int    Y = ++ X    // Y = 6    X = 6
```

如果运算符出现在变量的右边,则在赋值之后执行运算。下面的代码段中的 Y 的结果是 5。注意,在两种情况下 X 的值都是 6:

```
int    X = 5 ;
int    Y = X ++ ;    // Y = 5    X = 6
```

类似地减量操作符(- -)从变量上减 1,并且运算符放在变量的左边和放在右边所得的结果不同。

7.5　字符变量

Java 中的字符是一个特殊的集。用户即可将它当作 16 位的无符号整数,取值范围为 0 到

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

65535,也可以将它当作统一码字符。统一码标准已被许多不同语言的字母字符取代。拉丁字母、数字和标点符号在 ASCⅡ字符集(用在 PC 机上的集,值在 0～255 范围内)中都有自己的值。字符变量的缺省值是/u0000。

建立字符型变量的句法与整数型及布尔类型一样:

```
char myChar = 'b';
```

在这个例子中,myChar 变量被赋予了一个字符值'b'。注意在 b 的两边加单引号。这样做是告诉编译器 b 是字符值,而不是标识符 b。

7.6 浮点变量

浮点数是 Java 基本类型的最后一类。浮点数用于表示含有小数点的数字(如 5.3 或 99.234)。整数也可以用浮点数来表示,但作为浮点数的 5 实际上是 5.0。

在 Java 中,浮点数是用 float 类型和 double 类型表示的。这两种类型都符合标准的浮点规范:IEEE standard for Binary Floating-Point Arithmetic ,ANSI/IEEE std. 754-1985(IEEE, NewYork)。浮点数遵循这个规范意味着不管应用程序或小应用程序运行在什么系统上都不会遇到由浮点数差异引起的麻烦,这一点也增强了 Java 的可移植性。在其它语言中,符点运算是被定义为在运行程序的特定机器上的浮点单元。这就意味着在 IBM PC 上定义的 5.0 与 DEC VAX 定义的 5.0 不同。这两种浮点数的取值范围(不管在什么机器上都一样)如下表所示:

浮点类型	最小值	缺省值	最大值
float	1.40239846e-45f	0	3.40282347e+38f
double	4.94065645841246544e-324d	0	1.7976931348623157e+308d

另外,浮点数可以有四个特殊的状态:

- 负无穷大
- 正无穷大
- 零
- 非数

需要有这些状态是因为 754-1985 标准要求的,是出于对数字滚动的考虑。例如,对最大的浮点数执行加 1 运算将导致正无穷大。

许多在整数类型中可以执行的运算在浮点类型中也有类似的运算。主要的例外是位运算。表 7.5 列出了在浮点类型表达式中可以使用的运算符。

表 7.5 浮点表达式中的运算

运算	描述
=、+=、-=、*=、/=	赋值运算符
==、!=	等于和不等于运算符
<、<=、>、>=	不等于运算符
+、-	一元符号运算
+、-、*、/	加、减、乘、除运算符
+=、-=、*=、/=	加、减、乘、除并赋值运算符
++、--	增量和减量运算符

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

7.7　数组

有三种引用变量类型：

- 类
- 接口
- 数组

类和接口很复杂，所以它们各占一章单独介绍，但是数组比较简单，所以把它当作基本类型在这里介绍。

简单地讲，数组就是在一行中的几个项。如果有可以很容易索引的数据，那么使用数组来描述是再好不过了。例如，如果班级里有 5 个人，并且用户希望描述他们的智商，那么使用数组完全可以做到，下面就是这个数组：

```
int IQ[] = {123,109,156,142,131};
```

接下来是访问第三个人的智商的例子：

```
int ThirdPerson = IQ[2];
```

Java 中的数组不像大多数其它语言那么简单，它需要三步才能定义出数组，不是一步。

1.声明数组，有两种声明数组的办法。将一对方括号放在变量的后面，或者在标识符名后面。下面两行的效果一样：

```
int MyIntArray[];
int[] MyIntArray;
```

2.为数组分配内存空间。为了做到这一点，用户必须使用关键字 new，后面跟着变量类型和元素数量：

```
MyIntArray = new int[500];
```

3.在数组中放数据。对于本地类型(像本章中的那些数据)的数组，数组值最初被设置为 0。下一行代码显示了如何设置数组的第 5 个元素。

```
MyIntArray[4] = 467;
```

这时用户可能会问自己：为什么在 IQ 例子中能够创建有五个元素的数组并声明它们的值。在 IQ 例子中利用了捷径。仅就本地类型而言，用户可以通过在最初的声明行内将值放在大括号之内({})来为数组赋初值。

数组声明由下列部分组成：

数组修饰符	可选	用关键字 public、protected、private 或 synchronized
类型名	必须	构成数组的类型或类的名字
括号	必须	[]
初值	可选	在第 11 章中将详细介绍初始化
分号	必须	;

程序清单 7.3 显示了几个使用数组的例子。

程序清单 7.3　声明数组的例子

```
long Primes[] = new long[1000000];    // 声明数组并
                                      // 为其分配内存。
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
long[] EvenPrimes = new long[1];          // 这种情况也是数组。
EvenPrimes[0] = 2;                        // 为数组元素赋值。

//现在使用隐含式的'new'运算声明数组并为其赋值。

long Fibonacci[] = {1,1,2,3,5,8,13,21,34,55,89,144};

long Perfects[] = {6, 28};                // 创建一个包含两个元素的数组。

long BlackFlyNum[];                       // 声明数组。
                                          // 默认值为 null。

BlackFlyNum = new long[2147483647];       // 数组索引必须是整型值。

// 声明二维数组并为其赋值。
long TowerOfHanoi[][] = {{10,9,8,7,6,5,4,3,2,1},{},{}};

long[][][] ThreeDTicTacToe;               // 初始化三维数组。
```

还有几点关于数组的知识用户应当了解：

- 数组的索引是从 0 开始的(与 C 和C++ 一样)。换句话说,数组的第一个元素是 MyArray[0],而不是 MyArray[1]。
- 可以集中为数组赋初值。这只适用于本地类型。
- 数组索引必须是 int 类型(32 位整数)或者是能够强制为 int 的值。数组最多可容纳的元素数为 2,147,483,647 个。许多 Java 安装都达不到这个值,但它确实是语言定义的最大值。
- 当使用数组元素时,数组索引是括在方括号里的。

7.8 空白

使用空白对于许多语言是很重要的。空白把同一行的字符分隔开来,空白可由空格键、制表符、换行符或回车换行符产生。

在 Java 中空白可以用在不改变应用程序源代码的含义的情况下任意使用,只要不引起编译器误解即可。只有标志的中间这个位置不能加空白,比如变量名和类名,这是很明显的道理,因为下面两行代码的含义明显地不一样：

```
int myInt;
int my Int;
```

有些空白是可选的,但是适当地使用空白可以增加应用程序和小应用程序的源代码的可维护性。让我们看一下我们熟悉的 HelloWorld 应用程序,在这里,它使用了很少的空白：

```
public class HelloWorld{public static void main(String args[]){System.out.println("Hello World!!")}}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

很明显，这样的程序阅读起来比较费劲，每层意思的起止不明显，应该适当地使用有意义的空白将括号和程序语句调整位置使得结束花括号(})与和它相配的开始花括号({) 所在语句的句首对齐。

7.9　注释

注释可以作为提示帮助其它程序员阅读用户的程序代码的某些特殊部分。注释不是程序标志，也不是程序内容。

Java 支持三种注释风格：

- 传统(从传统的 C 语言继承来的)风格
- C++ 风格
- javadoc(在传统的风格上加了小小的修改)

7.9.1　传统的注释

传统的注释是指 C 语言风格的注释，即由斜杠星(/*)开始，以星斜杠(*/)结束。看一下程序清单 7.4，它显示了两个传统的注释。

程序清单 7.4　包含两个传统的注释的例子

```
/* The following is a code fragment
 * that is here only for the purpose
 * of demonstrating a style of comment.
 */

double pi = 3.141592654 /* close enough for now */;
```

正如用户看到的那样，这种注释可以是单行的，也可以是多行的。注释不能嵌套。如果用户试图嵌套它们，嵌套部分的开始不起作用，编译器认为它是注释的内容，而嵌套部分的结束却成了整个注释的结束。其后的文本将被解释成标记(即可执行部分)。程序清单 7.5 显示了这种注释法引起的混乱。

程序清单 7.5　看上去像两个注释的一个注释示例

```
/* This opens the comment
/* That looked like it opened another comment but it is the same one
* This will close the entire comment      */
```

7.9.2　C++ 风格的注释

第二种风格的注释是以双斜杠(//)开始并以本代码行结束为结束的。这种注释专门用来解释本行代码的含义。程序清单 7.6 显示了这种风格的注释的用法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 7.6 使用传统风格注释如C++风格的注释的例子

```
for (int j = 0, Boolean Bad = false;      // initialize outer loop
  j < MAX-ROW;                            // repeat for all rows
  j++) {
    for (int k = 0;   // initialize inner loop
    k < MAX-COL;  // repeat for all columns
    k++) {
          if (NumeralArray[j][k] > '9') { // > highest numeric?
               Bad = true;   // mark bad
          }                               // close if > '9'
          if (NumeralArray[j][k] < '0') { // < lowest numeric?
               Bad = true;      // mark bad
          }                               // close if < '0'
    }                                     // close inner loop
}                                         // close outer loop
```

7.9.3 javadoc 注释

这是 Java 中用到的最后一种注释。它具有前面提到的属性,但是这种注释的内容可以被 javadoc 工具使用,并自动地产生文档。

警告:如果用户打算使用 javadoc,那么应避免随意使用这种风格的注释。javadoc 程序无法区分这种不同。

javadoc 注释是以/ * * 开始以 * /结束的。以适当的方式使用这种注释,可以借助于 javadoc 自动地生成与 Java API 的文档页类似的文档页。程序清单 7.7 显示了一个 javadoc 的注释。

程序清单 7.7 javadoc注释的示例

```
/ * * This class is for processing databases
   * Example use:
   * xdb myDB = new xdb (myDbFile);
* System.out.println(xdb.showAll()); * /
```

7.10 源程序中的符号:赋值

前面介绍的布尔变量赋值只有两种可能的值:true 和 false 。对于整数,它几乎可以说有无数个可选值。另外可以使用多种方法为整数变量赋值。

最简单的整数赋值办法是使用传统的阿拉伯数字赋值:

```
in j = 3 ;
```

当使用其它形式的数字如十六进制的数字为整数变量赋值时将是怎样的情形? 为了告诉

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

计算机用户正在为变量赋一个十六进制的数字，用户需要使用代表十六进制的字符　对于十进制的 3 与十六进制的 3 没有什么区别。但是对于十六进制的 11，它的值是 17！用户一定要使用让计算机能够得到正确的值的方法来为变量赋值。

下面的语句中包含有各种代表数据格式的字符：

```
int        j = 0;
long       GrainOfSandOnTheBeachNum = 1L;
short      Mask1 = 0x007f;
static     String FirstName = "Ernest";
static     Char TibetanNine = '\u1049';
Boolean    UniverseWillExpandForever = true;
```

很明显，上面内容中包含了几种代表数据格式的字符。事实上 Java 语言中包含有五种主要的数据格式字符：

- 布尔
- 整数
- 字符
- 串
- 浮点数

7.10.1　代表整数格式的字符

代表整数格式的字符用于表达整数类型的数字。因为整数可以表示为十进制数(按 10 进位)、八进制(按 8 进位)或十六进制数(按 16 进位)，每种格式都有自己的数据格式字符。另外整数还可以使用可选的大写 L 或小写的 L(l)，L 位于数字之后，L 告诉计算机将该数字当做长(long)型整数(64 位)来处理。

和 C 与 C++ 一样，Java 中的十进制数可以是以任意非零数字字符开头的数(比如以 1 到 9 之间的任一个数开头)。八进制的数是以 0 开头的(045 与十进制的 37 相同)；在八进制的数里不使用 8 和 9 这两个数字字符。十六进制的数是用“0x”开头。十六进制的数是以由 0 到 9 的数字字符外加由 A 到 F 的英文字母(大写小写都可以)组成的。

最大的整数值和最小的整数用三种格式表示如下

```
最大的 32 位整数      2147483647
                      017777777777
                      0x7fffffff
最小的 32 位整数      -2147483648
                      020000000000
                      0X8000000
最大的 64 位整数      9223372036854775807L
                      0777777777777777777777
                      0x7fffffffffffffff
最小的 64 位整数      -9223372036854775808L
                      01777777777777777777777L
                      0xffffffffffffffffL
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

警告:试图表达超出上述表中数值范围的数将导致编译时刻错误。

7.10.2 代表字符格式的字符

字符格式符是单引号。不论字符值是拉丁字符、逃逸序列或其它任意的统一码字符都可以作为字符使用。单个字符可以是除(-)和(\)以外的任意可打印的字符。例如:'a' 'A' '9' '+' '-' 和 '~'。

某些字符,比如空格符很难像前面那样写出来,为了解决这个问题,这些字符使用逃逸序列字符来表示。逃逸序列字符是使用 '\b'格式表示的。单引号和反斜杠后面可以加下列内容:

- 另外一个字符(b、t、n、f、r、"、'或 \)
- 八进制数字序列
- u 和用于表示非行终结的统一编码字符的数字序列

逃逸序列如表 7.6 所示。

表 7.6 逃逸序列字符

逃逸序列字符	含 义
'\b'	\u0008 退格
'\t'	\u0009 水平制表符
'\n'	\u000a 换行
'\f'	\u000c 表格换行符
'\r'	\u000d 回车返回
'\"'	\u0022 双引号
'\''	\u0027 单引号
'\\'	\u005c 反斜杠

警告:不要使用 \u 格式来表达行结束,应该使用 \n 或 \r 字符。

表 7.6 中提到的字符格式符也叫做八进制逃逸字符。它们可以用于表达从 '\u0000'到 '\u00ff'之间传统的 ASCⅡ范围内的任意统一编码值。在八进制(按 8 进位)中的取值范围是 \000 到 \377。注意八进制数中只使用 0 到 7 之间的数字字符。表 7.7 显示了一些八进制字符的示例。

表 7.7 八进制值和它们的含义

八进制值	含 义
'\007'	\u0007 喇叭发声
'\101'	\u0041 'a'
'\141'	\u0061 'a'
'\071'	\u0039 '3'
'\042'	\u0022 '双引号'

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

警告:前面表中的字符格式符在 Java 编译器 javac 中是优先解释的,如果在程序中使用逃逸字符表示行结束符,比如回车或换行将导致在结束的单引号之前就结束了本行。这将导致一个编译时刻错误。回车和换行符在表 7.6 解释过。

7.10.3　代表浮点格式的字符

浮点数可以使用多种方式表示。下面是所有合法的浮点格式:

1003.45　　.00100345e6　　100.345E + 110345e - 2

1.00345e3　　0.0100345e + 6

浮点格式字符由几个部分组成,表 7.8 列出了这些部分及其使用规则:

表 7.8　浮点数的要求

部分	是否必须?	示例
整数部分	如果有小数部分它就不是必须的	0、1、2、……9 12345
小数点	如果有小数部分或指数部分它就是必须的	
小数部分	如果没有小数点就不需要出现。如果没有整数部分则必须出现	0、1、14159、718281828、411421、9944
指数部分	只有在没有小数点的情况下必须出现	e23、E - 19、E6、e + 307、e - 1
类型后缀	不是必须的。在没有类型后缀的情况下数值都是双精度的	f、F、d、D

整数部分不一定是单个数字符号;代表指数开始的 E 的大写与小写并不重要。F 和 D 都指的是类型。因此同一个数值可以有几种不同的表示法。

注意:如果非零的单精度数的绝对值超出了 1.40239846e - 45f 到 3.40282347e + 38f 范围将产生一个编译时刻错误。非零双精度数绝对值的范围是 4.94065645841246544e - 324 到 1.7976931348623157e + 308。

7.10.4　代表串格式的字符

字符串不是真正的本地类型。但在这里还是通过介绍字符串来结束对代表格式的字符的讨论。字符串是由带有双引号的零个或多个字符组成的。这些字符可以是在本章前面"代表字符格式的字符"部分的逃逸序列中列出的字符。双引号必须出现在源代码的同一行,所以字符串中可以直接包含换行字符。为了达到换行的目的,用户必须使用逃逸序列如 \ n 或 \ r。

双引号(")字符和反斜线(\ )字符也必须使用逃逸序列表示(\ "和 \ \ )。

一个从C++继承来的漂亮特性是:如果用户需要使用较长的串,可以先建几个较小的串,然后再用加号(+)把它们加在一起。

警告:虽然在字符串操作中使用 + 运算符可以提供一些方便,但对于 String 类可能就不是非常高效的。因为字符串连接毕竟要浪费内存资源。

下面是一些字符串示例:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
"Java"
"Hello World ! \ n"
"The Devangari numral for 9 is \ u096f"
"Do you mean the European Swallow or the African Swallow ?"
"* * * * ERROR 9912 Data/Time 1/1/1900 00:01"
+ "predates last backup : all files deleted !"
"If this were an actual emergency"
```

7.11 创建和撤销对象

对于计算机语言来讲，内存管理是一个非常重要的课题。不论何时为类创建的实例，Java运行时刻系统都为它开辟一部分内存空间以便存储用于管理这个实例的信息。但是当这个对象出了程序范围或不再被程序使用时，这部分内存将被释放掉并交由其它对象使用。

虽然Java对程序员隐藏了许多操作，但也提供了通过执行一定的附加任务来优化用户的代码的机会。虽然Java要求用户显式地使用new操作符为每个对象分配内存，但也允许指定新对象(通过使用构造器方法)和当撤除对象时彻底地撤出内存。

> **注意**：与C和C++不同，Java为用户提供了大量的内存管理控制，Java可以为用户执行许多这种任务。最值得注意的是内存无用单元回收，Java通过它自动地将不再引用的对象释放掉，因此使得C++的free( )方法不再是必要的了。

7.11.1 使用new操作符创建对象

当创建类的实例时，开辟一块内存用于存放该对象的数据是必须的，当在类的开始声明实例时，用户仅仅告诉编译器使用某个名字的变量将在该类中被用到，但并没有真正地为它分配内存。紧接着必须为该变量使用new操作符分配内存空间。请考察一下下列代码：

```
public class Checkers
{
private GameBoard board ;
public Checkers ( ){
board = new GameBoard ("Checkers")
board.clean Board ( ) ;
}
```

可以看到，虽然变量board在第三行中声明了，但用户必须使用new操作符为它分配内存。使用new操作符的句法是：

类的实例 = new 类名（可选的参数）；

非常简单，该行告诉编译器为该类的实例分配内存并将变量指向这部分内存。在这个过程中编译器也调用类的构造器方法并将适当的参数传给它。

指针：事实或虚拟？

Java声明不再支持指针，目的是为防止程序员发生与指针处理有关的错误。虽然Java采

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用了不使用基于指针的思维模式，但 Java 仍然要不得不处理相同的内存分配问题和引用这些内存地址。

虽然赋予了不同的名字，但引用仍是 Java 中指针的翻版。用户不使用指针执行侵略性操作，但仍需要进行指针赋值和对象创建。用户必须首先声明变量(引用)。然后必须为它分配适当的内存和为它指定引用。因为用户可能后来决定为其它类型的相同变量(或 null)设置引用，Java 的引用系统与 C 的指针引用系统非常类似。

虽然 Java 的实现为程序员有效地隐藏了指针行为并避免了用户发生这一类错误，但是，考虑在用户创建和引用变量时后台发生了什么事情是一个良好的动机。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第8章 方 法

本章内容：

8.1 方法的两个部分

8.2 块和语句

8.3 分隔符

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

8.1 方法的两个部分

方法是 Java 程序的心脏和灵魂。Java 中方法的用途与 C、C++、Pascal 里的函数一样，小应用程序和应用程序中的所有动作都是由方法完成的，只有把各种动态的方法有机结合起来才能编写出大规模的 Java 应用程序。

就像 C 和 C++ 的函数一样，Java 的方法是类的本质所在，它负责管理要执行的任务。方法包括两个部分：声明和体。虽然方法的实现实际上是包含在方法的体中，但是一些重要的信息是包含在方法声明中。

最简单的方法(也是最没有用的方法)是下面这个方法：

```
void SimpleMethod ( ){
}
```

8.1.1 声明

方法的声明与前面的代码中的第一行类似。方法声明中至少要指明方法的返回值类型和方法的名字。一般来说除了这两个必要的项目以外，方法声明中还可以有其它的选项，在下面将看到这些选项。方法声明的完整格式是：

访问说明符 修饰符 返回值 方法名 (参数) throws ExceptionList

在这里斜体部分是可选的。

8.1.1.1 访问说明符

方法的第一个选项是访问说明符。访问说明符用于限制对方法的访问。不管使用什么样的访问说明符，同在一个类中的其它方法都可以访问该方法。虽然同一个类中的方法都可以被其它的类访问，但是有些必要的任务用户可能不希望被其它对象执行。在第 11 章“类”中用户可以了解更多与类有关的知识。现在用户先来看一下访问说明符如何修改方法。

Public public 说明符是方法中最松散的说明符。通过将方法指定为 public，可以使得该方法能够被世系的所有类或包中的所有类访问。换句话说 public 方法不受任何限制。例如：

```
public void toggle Status ( )
```

Protected 第二个访问说明符是 protected。protected 方法可以被由当前类扩展出来的任何类访问。例如，java.awt.Conponent 类有一个 protected 方法名叫 paramString()，在 java.awt.Button 类中可以使用它，因为 java.awt.Button 类是由 java.awt.Component 类扩展来的，但在不是由 java.awt.Component 扩展出来的任何类中是不能访问该方法的。例如：

```
protected void toggleStatus ()
```

参见第 11 章的“使用包来组织代码”部分。

> **注意**：如果在编译程序时出现试图越界访问错误，用户应当对程序进行错误查找。因为错误信息并不会告诉用户是因为试图访问 protected 方法引起的错误。相应的错误信息可以是：

No method matching paramString() found in class java.awt.Button。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在 java.awt.Button 中，java.awt.Button.paramString() 是受保护的方法。这是因为受限制的方法被有效地对无权类隐藏了起来。所以当编译不符合安全性限制的方法时，这些方法就从编译器中隐藏掉了。

还应当注意当在访问权范围之外访问私有的和保护的方法时会遇到类似的错误，当从一个未授权的类中访问某个受保护的域同样也会出现错误。

Private private 是方法中保护程度最高的说明符。

私有方法只能被同一个类的方法调用。即使是由本类扩展的类也不能访问该私有类。下面是私有类的例子：

```
private void toggleStatus ()
```

Default 这种方法不把类声明为在 Java 中具体的访问类型，这种方法可以被当前包中其它的任何类访问，但是不允许当前包以外的类访问。下面是一个例子：

```
package abc ;
public class NetworkSender {
  void sendInfo (String mes ){
    system.out .println(mes)
  }
}

package abc ;
public class NetworkSender Test {
  String mes = "test";
  void informOthers(String mes){
    NetworkSender messenger ;
    messenger = new NetworkSender() ;
messager.sendInfo(mes) ;//这是合法的
}
}

package xyz ;
import NetworkSender ;
public class NetworkSenderTest2 extends NetworkSender {
  String mes = "test";
  void informOthers(sering mes){
    NetworkSender messenger ;
    messenger =new NetworkSender();
    messanger.sendInfo (mes );//这是非法的
  }
}
```

程序中的那个合法的语句调用了属于方法 NetworkSender 方法的 sendInfo()方法，这是合法的，因为 default 方法可以被同一个包中的其它类访问（NetworkSender 和 NetworkSenderTest 都是 abc 包的。在 NetworkSenderTest2 中那个不合法的语句是因为它试图在 NetworkSender 类的实例中调用 sendInfo()方法，但是 NetworkSenderTest2 是在不同的包中的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

(xyz 包)。即使 NetworkSenderTest2 是 NetworkSender 的子类，它也只算是引用 Network-Sender 实例的 sendInfo()方法，而不是引用属于超类的方法。

8.1.1.2　修饰符

方法修饰符允许用户为方法设置属性，比如它在何处可见，当前类的子类如何与它交互。

Static　static 或类、变量和方法是紧密相联的。例如：

static void toggleStatus ()

区分类的特定实例的属性和类本身是非常重要的。在下面的代码中(见程序清单 8.1)，创建了两个 Elevator 类的实例并使用它们执行同一个操作。

程序清单 8.1　Hotel.java——带有方法实例的饭店示例

```
class Elevator {
    boolean running = true;
    void shutDown() {
        running = false;
    }
}

class FrontDesk {
    private final int EVENING = 8;
    Elevator NorthElevator, SouthElevator;

    FrontDesk() {                              // the class constructor
        NorthElevator = new Elevator();
        SouthElevator = new Elevator();
    }

    void maintenance(int time) {
        if (time == EVENING)
            NorthElevator.shutDown();
    }

    void displayStatus() {
        // code is very inefficient, but serves a purpose
        System.out.print("North Elevator is ");
        if (! (NorthElevator.running ))
            System.out.print("not ");
        System.out.println("running.");
        System.out.print("South Elevator is ");
        if (! (SouthElevator.running ))
            System.out.print(" not ");
        System.out.println("running.");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
}

public class Hotel {
    public static void main(String args[]) {
        FrontDesk lobby;
        lobby = new FrontDesk();
        System.out.println("It's 7:00. Time to check the elevators.");
        lobby.maintenance(7);
        lobby.displayStatus();

        System.out.println();
        System.out.println("It's 8:00. Time to check the elevators.");
        lobby.maintenance(8);
        lobby.displayStatus();
    }
}
```

NorthElevator 和 SouthElevator 都是 Elevator 类的实例。这意味着它们都创建了自己的 running 变量和 shutDown()方法。从程序中可以看到，虽然最初它们是同样的电梯，当 maintenance()方法一经调用，NorthElevator 的 running 状态就与 SouthElevator 的 running 状态不同了。

接下来，如果编译并运行这些代码，将产生下列输出结果：

```
c: \ dev > \ JDK1.2 \ java \ bin \ java Hotel
It's 7:00. Time to check the elevators.
North Elevator is running .
South Elevator is running .
It's 8:00 Time to check the elevators .
North Elevator is not running .
South Elevator is running .
```

注意：前面的例子中，用户可能注意到看上去有些滑稽的方法 FrontDesk()了。

它是什么？用户将在第 11 章的"构造器部分"里了解到，FrontDesk()是 FrontDesk 类的构造器。无论何时调用 FrontDesk()方法都将创建一个 FrontDesk 的实例，构造器为用户提供初使化域和执行其它的这类准备性操作的能力。

像 running 和 shutDown()这样的变量和方法叫做实例变量和实例方法。因为每当 Elevator 类被用实例说明时，就创建这两个实例的拷贝。在前面的例子中，running 变量的值一定是可以改变的，因为它们是两个拷贝，改变其中一个拷贝里变量的值不会影响另一个拷贝中的相应变量的值。所以用户可以分别地跟踪北面电梯(North Elevator)的状态和南面电梯(North Elevator)的状态。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果用户希望定义或修改所有电梯(elevator)的某个属性该怎么办？现在让我们考查下程序清单 8.2 就可以得到定义或修改的办法，请注意程序清单 8.2 中的新增加部分。

程序清单 8.2　Hotel2.java——带有静态方法的饭店示例

```
class Elevator {
    boolean running = true;
    static boolean powered = true;
    void shutDown() {
        running = false;
    }
    static void togglePower() {
        powered = ! powered;
    }
}

class FrontDesk {
    private final int EVENING = 8;
    private final int CLOSING = 10;
    private final int OPENING = 6;
    Elevator NorthElevator, SouthElevator;
    FrontDesk() {
        NorthElevator = new Elevator();
        SouthElevator = new Elevator();
    }

    void maintenance(int time) {
        if (time == EVENING)
            NorthElevator.shutDown();
        else if ( (time == CLOSING) || (time == OPENING) )
            Elevator.togglePower();
    }

    void displayStatus() {
        // Code is very inefficient, but serves a purpose.
        System.out.print("North Elevator is ");
        if (! (NorthElevator.running ))
            System.out.print("not ");
        System.out.println("running.");
        System.out.print("South Elevator is ");
        if (! (SouthElevator.running ))
            System.out.print(" not ");
        System.out.println("running.");
        System.out.print("The elevators are ");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        if (! (Elevator.powered))
            System.out.print("not ");
        System.out.println("powered.");
    }

public class Hotel2 {
    public static void main(String args[]) {
        FrontDesk lobby;
        lobby = new FrontDesk();
        System.out.println("It's 7:00. Time to check the elevators.");
        lobby.maintenance(7);
        lobby.displayStatus();

        System.out.println();
        System.out.println("It's 8:00. Time to check the elevators.");
        lobby.maintenance(8);
        lobby.displayStatus();

        System.out.println();
        System.out.println("It's 10:00. Time to check the elevators.");
        lobby.maintenance(10);
        lobby.displayStatus();
    }
}
```

在这个例子中，变量 powered 是一个静态变量，方法 togglePower()是一个静态方法。这就意味着它们是所有 Elevator 类的属性，而不只是特定实例的属性。调用 NorthElevator.togglePower()方法、SouthElevator.togglePower()方法或 Elevator.togglePower()方法都将改变类中所有的 powered 变量的状态。

接下来编译并运行该代码。代码生成的输入结果如下所示：

```
c: \ dev > \ JDK1.2 \ java \ bin \ java Hotel2
It's 7:00. Time to check the elevators.
North Elevator is running .
South Elevator is running .
The elevators are powered .
It's 8:00. Time to check the elevators .
North Elevator is not running .
South Elevator is running .
The elevators are powered .
It's 10:00. Time to check the elevators .
North Elevator is not running .
South Elevator is running .
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

The elevators are not powered .

在方法声明的前面放置 static 使得方法成为静态的方法。非静态的方法也可操作静态的变量,而静态的方法只能处理静态的变量和静态的方法。

Abstract　抽象方法是被简单地声明了一下的方法,但在当前的类中没有定义方法的体。这个责任留给了当前类的子类。下面是定义 abstract 的例子:

abstract void toggleStatus ()

警告:静态类和类构造器都不能被声明为抽象的(abstract)。也不能对下面将介绍的 final 使用 abstract 方法,因为禁止这种用法可以使得用户避免覆盖该方法。

Final 将关键字 final 放在方法声明的前面,可以避免当前类的子类覆盖已有的方法。这种能力提高了类的隔离程度,这样可以保证用户在方法中定义的功能永远不会以任何方式改变。例如:

final void toggleStatus ()

Native　native　方法是用户在 Java 中可以使用但是不能写的方法。

通常 native 方法是用 C++ 编写的,它有几个好处,比如运行速度快。像抽象方法一样,可以通过将修饰符 native 放置在方法声明的前面来声明它们并使用分号来代替方法体。

记住告诉编译器方法的属性的声明是很重要的。因为用户在类中指定与源代码中的返回类型和使用的参数一致的返回类型和参数是至关重要的。

Synchronized　可以通过将关键字 synchronized 放置在方法声明之前来声明某个方法是 synchronized 型的。使用同步(synchronized)修饰符可以在两个方法同时试图访问某一块数据避免数据毁损。虽然在简单的程序里不涉及这个问题,但是当用户在程序中使用多个线程时,这可能会变成一个严重的问题。下面是 synchronized 的例子:

synchronized void toggleStatus ()

参见第 13 章的"什么是线程"部分。

8.1.1.3　返回信息

虽然返回信息是程序能够干的事情中最重要的事情,但在这里仅就返回信息的某一方面进行详细讨论。Java 的方法可以返回任何一种数据类型,简单的如整数和字符,复杂的如对象(这意味着也可返回字符串,浮点数等类型)。

请记住:除非使用了关键字 void 用作返回类型,用户必须在声明中指定返回变量的类型。

例如,在下面的方法中声明返回布尔型变量。返回值实际是由第三行 return true;语句和第四行 return false;语句完成的。

8.1.1.4　方法名

与方法命名有关的规则非常简单,它与所有其它的 Java 标识符命名的规则完全一样:以统一码字符(或下划线或货币符号)开头后面只跟统一码字符。

8.1.1.5　参数表

参数表是将要传递给方法的信息列表。它可以是下面的格式并且可以由多个参数组成:

数据类型　变量名,　数据类型　变量名,……

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:如果没有参数要传递,那么 Java 允许参数为空(这一点与其它语言不同,它们不允许缺略参数表,比如 C 语言,C 语言要求关键字 void)。因为当不使用参数时就像下述语句一样:

```
public static final void cleanBoard ()
```

在 Java 中传递参数

在 C 语言和 C++语言中,变量总是要传递值的。在 pascal 中引用总是要传递变量的。但在 Java 中,这要取决于用户使用的数据类型了。这可能是整个 Java 语言中最模棱两可的部分。这里有个规则:如果被传递的是基本类型(如 int、char 或 float),那么结果就使用值来传递。如果传递的类型是对象(如用户创建的类),那么对象就由引用来传递。

这是什么意思?如程序清单 8.2 所示,如果用户传递 int 给方法,那么方法就修改这个 int,在原来的类中 int 还是原来的值。但是当类被传递时,原来的方法中变量的值也跟着被修改。看一下程序清单 8.3。

程序清单 8.3 passingDemo——传递对象与传递基本类型的结果

```
public class passingDemo {

    public void first(){
        xObject o = new xObject ();
        o.x = 5;
        int x = 5;

        changeThem (x, o);
        System.out.println();
        System.out.println("Back in the original method");
        System.out.println("The value of o.x is " + o.x);
        System.out.println("But, The value of x is now " + x);
    }

    public void changeThem (int x, xObject o){
        x = 9;
        o.x = 9;
        System.out.println("In the changThem method");
        System.out.println("The value of o.x is " + o.x);
        System.out.println("The value of x is now " + x);
    }

    public static void main(String args[]){
        passingDemo myDemo = new passingDemo();
        myDemo.first();
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
}

class xObject {
        public int x = 5;
}
```

上述代码的输出结果是：

```
In the change Them method
The value of o.x is 9
The value of x is 9

Back in original method

The value of o.x is 9
The value of x is 5
```

引用传递或值传递 理解任何编程语言的一个重要的事情是：传递给方法的值是值还是引用。如果语言使用引用传递，当用户将值作为参数传递给方法然后再修改这个值会引起调用方法中该变量值的改变。而另一方面，如果使用值传递，只有传递给方法的值发生改变，但在调用方法中不出现任何改变。

Java实际上是一种混合系统。基本类型(byte、short、char、int、long、float、double 和 boolean)是值传递。所有的对象是引用传递。这就是为什么在前面的例子中，原方法中的 x 的值不发生变化的原因。另一方面由于 o(一个对象)是引用传递，所以当在原方法中打印 o.x 时，它的值发生了变化。

8.2 块和语句

Java中的方法和静态初始化符是由语句块定义的。语句块是括在花括号{}中的语句系列。当语句的构成将一个语句或子语句作为一部分调用时，可以在子语句的位置上插入一个块。

最简单的块{}如下面的例子所示：

```
public void Hithere(){
}
```

下面的例子稍微复杂一点：

```
public void HiThere{
    int Test ;
    Test = 5 ;
}
```

代码块不仅仅是用于定义方法的开始和结尾的，它也可以在整个代码的任意位置上使用。块的一个非常重要的方面是语法上可以当做一个指令，这就意味着可以把大的代码块放在一起当做一个指令行来执行。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java 不禁止程序员将代码分成若干块。甚至可以编写一些根本就不调用的块。下面的代码片段演示了这种合法的但不常用的技术：

```
String Batter ;
Short Inning , Out ,Strikes ;
Batsman     Casy ; // Batsman 的对象
.....
If [(Inning = =9) & & (Out = =2) & & (Batter.equals ("casey"))]{
Casey.manner("ease");
Casey.bearing ("pride");
}          //新块开始
int OnlyExistsInThisBlock =1;
Casey.face ("smile")
Casey.hat ("lightly doff")
//块结束
```

请注意，这个程序片段包含有两个完整的块。一个是 if 语句的子语句 另一个是没有必要的块，它包含了无用的整数 onlyExistsInThisBlock。

8.2.1 语句加标志

在 Java 中任何语句都可以拥有标志，实际拥有与其它标识符一样的属性；它不能与关键字或已经声明为本地标识符的字串同名，如果它与变量，方法或类型同名，那么在这个块中，新的标志优先，而块外面的变量、方法或类型将被隐藏。标志有当前块的范围。标志后面跟有冒号。

标志只是被 break 语句和 continue 语句使用。

下面的代码片段中有带标志的语句：

```
writhing:
    Pitcher.Grindsball ("Hip")
    Casey .eye("Defiance Gleams")
    Casey .lip ("Curling Sneer")
pitch:while (strike ++ <2){
    if (strike <2) continue pitch ;
    break writhing ;
}
```

writhing 语句是一个标志表达式语句，在这段程序里方法是从更复杂的 pitch 对象中调用的。pitch 语句是加了标志的循环语句(while)。这个标志是被当作 continue 语句的参数来使用的。

8.2.2 范围

块的另一个用途是控制对象的范围。当用户在块中声明变量后，该变量只在给定的代码块中使用。例如，有下面这样的块：

```
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    int x = 5
}
System.out .println("x is =" + x) ; //这行不能用。
```

代码的最后一行是无效的，因为计算机创建了 x 变量，但遇到了右括号(})，所以就把 x 给扔掉了。

8.3 分隔符

分隔符是单字符的标记，分隔符(就像它的名字暗示的那样)出现在其它标记之间。Java 有 9 个分隔符，大概地描述如下：

(用于打开方法的参数列表和在表达式中建立操作的优先级。

) 用于关闭方法的参数列表和在表式式中建立操作的优先级。

{ 用于打开语句块和初始化列表。

} 用于关闭语句块和初始化列表。

[放在表达式前面用作数组索引。

] 跟在表达式后面用作数组索引。

; 用在表达式语句末尾和分隔 for 语句的各部分。

, 用于分隔列表中的内容。

. 用作小数点和分隔包名与类名及方法名或变量名等。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第9章　使用表达式

本章内容：

9.1　什么是表达式
9.2　如何计算表达式
9.3　Java 表达式与 C 语言表达式的差异
9.4　位运算符
9.5　移位运算符
9.6　类型转换
9.7　字符串加法

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

9.1　什么是表达式

表达式是运算符与操作数的结合。像其它语言一样，表达式也是 Java 语言重要的构造块。表达式允许用户执行数学计算、串联接，比较值、执行逻辑运算和操作对象。没有表达式的编程语言是沉闷的，用处不大和没有生命力的。

在前面几章里用户已经见过一些表达式了，在本书的其它章中，大部分表达式都是比较简单的，在第 7 章"数据类型及其它标记"中着重介绍了运算符，运算符是表达式中两个关键元素之一，也是 Java 主要标记之一，像关键字注释等都是 Java 标记。在本章中用户应仔细观察如何使用运算符建立表达式，换句话说，如何让运算符按用户的意思工作。

这里列出了各种关于表达式是什么的技术定义，但最简单的定义是：表达式是运算符与操作数结合的产物。表达式通常用于对变量或值执行运算和操作。在表 9.1 中用户可以看到几个合法的 Java 表达式。

表 9.1　合法的 Java 表达式

表达式名	例子
加法表达式	X + 5
赋值表达式	X = 5
数组索引	sizes
方法调用	Triangle. RotateLeft(50)

9.2　如何计算表达式

当表达式简单时，像表 9.1 中列出的表达式，计算表达式的结果是很容易的事情。当表达式比较详细并且有多个运算符时，事情就变得复杂了。

正如用户在第 7 章中学习到的那样，表达式就是运算符和操作数的结合。虽然表达式定义是正确的，但却不是非常有帮助的。有时用户需要建立和使用非常复杂的表达式——可以执行某种复杂的计算或其它操作。为了做到这一点，用户需要深入理解 Java 表达式是如何创建和计算的。在本部分中用户可以看到帮助用户使用表达式的三个主要工具：运算符关联、运算符的优先顺序和计算顺序。

9.2.1　运算符关联

最容易的表达式规则是关联。所有的算术运算符都是从左到右关联的；这就意味着如果在表达式中同一个运算符出现多次，如 a + b + c 中的加号，那么最左面的运算符首先计算，然后逐个向右移。考虑下面的赋值语句：

```
x = a + b + c ;
```

在这个例子中，等号(=)右边的表达式的值经过计算后被赋给左边的变量 x。在计算右边的值时，+ 运算符关联从左到右意味着 a + b 的值首先计算，然后把结果加到 c 中。第二次计算结果就是要赋给 x 的值。如果明确地用括号写出计算的顺序，那么表达式将是下面这个样子：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
x = [(a + b) + c];
```

注意:在前面的例子中,a + b + c 表达式中同一个运算符出现了两次。只要同一个运算符出现不只一次,用户都可以使用从左到右的关联规则。

用户应当使用关联规则来计算下列赋值语句右边的表达式的值:

```
volume = length * width * heigt;
OrderTotal = SubTotal + Freight + Taxes ;
PerOrderPerUnit = Purchase/Orders/Units ;
```

在这些表达式中只有最后一个表达式在错误的关联情况下将得出不同的结果。这个表达式的正确答案是

```
(Purchase/Order)/Units
```

而不正确的关联的错误结果是:

```
Purchase/(Orders/Units)
```

它也可以被写成

```
(Purchase * Units )/Orders
```

很明显,它与正确的表达式的结果是不一样的。

9.2.2 Java 运算符的优先顺序

当用户有一个包含不同的运算符的表达式,那么关联规则就不适用了,因为关联只帮助确定要被计算的相同的运算符的组合。现在用户需要知道使用不同运算符的表达式如何计算。

优先级帮助用户确定哪个运算符首先执行。如果用户写一个 A + B * C 表达式,根据数学规则,用户将首先计算 B 乘 C,然后把结果加到 A 里。优先级帮助计算机做同样的事情。乘除运算符(*、/和%)具有比加减运算符(+和-)更高的优先级。所以,在由加减运算符和乘除运算符混合组成的表达式中,首先计算乘除运算符。

考虑一下下面的赋值语句,它试图将华氏温度转换为摄氏温度。

```
Celsius = Fahreheit - 32 * 5/9
```

正确的摄氏温度等于华氏温度减去 32 再乘以 5 除以 9。在这个等式中,因为 * 运算符和 / 运算符有较高的优先级,所以先计算子表达式 32 * 5/9(得出结果 17),然后再从 Fahrenheit 变量中减去 32 * 5/9 的结果。

为了正确地书写这个等式,用户需要改变表达式的计算顺序,为此用户可以使用括号。括号内的任何表达式都是要首先执行的。为了执行前面例子中的正确转换,可以使用下面的表达式:

```
Celsius = (Fahrenheit - 32) * 5/9 ;
```

注意:有趣的是某些计算机语言不使用优先顺序规则。某些语言,像 APL,它直接使用从左到右或从右到左的计算顺序,不管包含的是什么运算符。

下面的例子也是借助于括号实现的:

```
NewAmount = (Saving + Cash) * ExchangeRate ;
TotalConsumption = (Distance2 - Distance1) * ConsumptionRate ;
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

一元算术运算符的优先级非常高；比其它所有的算术运术符的优先级都高。在下面的例子中 Ryman 的值是负 5 乘以 Xantham，而不是 5 乘以 Xatham 再取负值（尽管二者的结果相同）。

```
Ryman = -5 * Xantham;
```

9.2.3　小结——运算符表

表 9.2 是优先顺序表。优先级最高的在顶部，同一行的运算符具有相同的优先级。

所有这些运算符中除了一元运算符、赋值运算符和条件运算符外都是从左到右关联的。对于任何单 -的运算符，操作数计算是严格地从左到右的，并且所有的操作数在运算执行前都已经被计算了。

表 9.2　完整的 Java 运算符优先级表

描　　述	运　　算　　符
高优先级	. [ ] ( )
一元运算符	+ - ~ ! + + - -
倍运算运算符	* / %
加减运算符	+ -
移位运算符	《 》 > > >
关系运算符	< < = > = >
等于运算符	= = ! =
按位或运算符	& ^
条件与运算符	|
条件或运算符	& & | |
条件运算符	?:
赋值运算符	= op =

9.2.4　计算顺序

许多人在第一次学习计算机语言时，总是把运算符优先级和计算顺序搞混淆。这两个概念实际是不同的。优先级规则帮助用户确定表达式中哪个运算符在先和哪些操作数执行该运算符的运算。例如，在下面的代码行中 * 运算符的操作数是 a 和(b + c)；

```
d = a * (b + c);
```

计算规则的顺序不是决定何时进行运算符定义的运算，而是决定何时计算操作数。

这里有三个帮助用户记住表达式如何被计算的规则：

- 对于二元运算符，左边的操作数先计算，右边的操作数后计算。
- 操作数总是在执行运算符定义的运算之前计算出来的。
- 如果在方法中调用了多个参数，这些参数是用逗号隔开的，那么参数是严格按照从左到右的规则的进行计算的。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

9.3 Java 表达式与 C 语言表达式的差异

因为 Java 是 C 和C++ 的进化，这三种语言中的表达式语法非常相似。如果用户使用过 C 语言，那么记住这三种语言仅仅是相似，但不是一致是很重要的。

一个非常重要的不同是在 Java 中计算顺序是受保证的，而在 C 语言中一般是未定义的或取决于计算机系统的。在 Java 中，取余运算符(%)、增量运算符(++)和减量运算符是为所有的基本数据类型(除布尔运算符之外)定义的；而在 C 语言中，仅仅是为整数定义的。

在 Java 中关系运算符和等于运算符产生布尔型结果，而在 C 语言中它们产生 int 型结果。Java 中的逻辑运算符被严格限制在布尔型中使用。

Java 支持字符串的本地运算，包括串联接和串赋值。而 C 语言对字符串没有这些支持。

在 C 语言中，对有符号数使用右移将导致由具体的计算机系统决定的行为。Java 为了避免这种混淆使用了两种右移运算符，一种是补零，另一种是补符号位的值。

9.4 位运算符

如果有一个数，比如是 0x0F2(它是十六进制的数，等于 242)，如何将其中的 2 扔掉呢？又如何找出 0x0F2 与 0x0A1 相同的置位呢？使用位运算符可以很容易地解决这两个问题(答案是：0x0F2&0x0F0 和 0x0F2&0A1)。

位运算符是一些操作符的集，对用户来说它可能是非常重要的，也可能是无关紧要的，这取决于干什么。当需要使用位运算符时，很少能找到其它的运算符能如此容易地得到所需的结果。但是，大部分工作不需要执行这种神秘的计算。

什么是位运算符呢？位运算符是工作在计算机值存储的底层上的。数字是用一系列的开和关(也就是位)序列存储的，它一般被翻译成二进制的数 1 和 0；典型的变量像 int 由 32 个这种 1 和 0 构成的。能够直接操作这些值是很有帮助的，位运算符就是干这种工作的。

让我们看一个使用字节的简单示例。字节由 8 个内存位组成。这 8 个位每一个可以是 0 或 1，整个数的值由每个位以 2 为底的指数的和决定。最右面的位代表的是 0 或 1 接下来的位代表 0 或 2；再后的位代表 0 或 4 等等，每个位的值是 0 和 2^n，n 指的是位数。表 9.3 显示了几个数的二进制表示。

表 9.3 十进制数的二进制表示

十进制数	值	128	64	32	16	8	4	2	1
17	0	0	0	0	1	0	0	0	1
63	0	0	0	1	1	1	1	1	1
75	0	0	1	0	0	1	0	1	1
131	0	1	0	0	0	0	0	1	1

为了求出表 9.3 中二进制数相应的十进制值，需要把每个二进制数的包含 1 的列最顶层的数字加起来。比如，第一行的值是

16 + 1 = 17

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 9.3 中的数全部是正数，这样做是有目的的。负数的表示稍微困难些。在 Java 中对于任意整数，除了 char(字符)型外，最左边的位是为符号位保留的，如果符号位是 1，那么这个值为负。剩下的位是十进制补码，但是现在不用关心它。浮点数也有它们特殊的二进制表示法，但在本书中不加以讨论。

有三个二进制位操作符对每一位执行逻辑的 AND 运算、OR 运算和 Exclusive OR(有时叫 XOR 即异或)运算。这三个运算符是：

- 按位与：&
- 按位或：|
- 按位异或：^

每个运算符都产生一个基于真值表的结果。每个运算符都有一个真值表，下面的三个表显示了它们。

为了确定位运算的结果，需要看一下每个作为位集的操作数并将它的位与适当的真值表比较。

第一个值(A)	第二个值(B)	结果值(A&B)
0	0	0
0	1	0
1	0	0
1	1	1

第一个值(A)	第二个值(B)	结果值(A|B)
0	0	0
0	1	1
1	0	1
1	1	1

第一个值(A)	第二个值(B)	结果值(A^B)
0	0	0
0	1	1
1	0	1
1	1	0

位运算符的操作数可以是布尔型，当然也可以是其它任意的整数类型。

表 9.4 显示了两个例值在执行各种运算时的结果。用户首先看到的是两个数 11309 和 798 的布尔值。然后是使用各种位运算符后的结果。

表 9.4　位运算实例

表　达　式	二　进　制　表　示
11309	0010　1100　0010　1101
798	0000　0011　0001　1110
11309&798	0000　0000　0000　1100
11309|798	0010　1111　0011　1111
11309^798	0010　1111　0011　0011

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

9.5 移位运算符

在 Java 中有 3 个移位运算符，如下所示：

- 左移：<<
- 带符号右移：>>
- 无符号右移：>>>

移位运算符将数中的所有位向左或向右移动。运算符左面是要移位的操作数，而右面的操作数是要移动的位数。例如：

17 << 2

数字 17 将被向左移动两位。左移和无符号右移用零来填补空位。有符号右移使用符号位来填补空位。下面的表显示的是两个 8 位数 31 与 - 17 和当它们被移位后的结果。

数	x	x<<2	x>>2	x>>>2
31	00011111	01111100	00000111	00000111
- 17	11101111	10111100	11111011	00111011

移位运算符的优先级高于关系运算符，但是低于附加的算术运算符。

9.6 类型转换

通常，在各种类型中非常重要的一个方面是它们是如何相互联系的。换句话说，如果有一个像 1.2 这样的浮点数，那么它与整数有什么关系呢？当将一个字节型数(8 位)加到一个整形(32 位的 int)数时语言如何处理这种情况呢？为了解决这个问题，Java 执行了类型转换，Java 是强类型语言，因为在编译时必须知道每个变量的类型。当值从一种类型转向另一种类型时 Java 执行扩展的类型检查(帮助程序员检查错误)并设置严格的限制。

有两种不同的转换：

- 显式转换发生在用户慎重地改变值的数据类型时。
- 隐式转换发生在一个等式中有两个不同的类型时，通过隐式转换可以将它们调整一致。这种转换不需要用户指明，甚至发生在用户根本不知道的情况下。

简单地说，强制和转换是 Java 在表达式中使用不同数据类型的方法。

> **注意**：在 C 语言中，通过赋值语句几乎所有类型都可以转换成其它任意的类型。但在 java 中不是这样，java 中的隐式类型转换只在不损失精度和不改变量值的情况下才执行。否则会导致编译器错误。除非使用显式的强制。

9.6.1 隐式类型转换

在计算表达式时，Java 执行大量的隐式转换，转换的规则非常简单，并且有比 C 语言甚至是 C++ 更多的控制。

对于一元运算符(比如 + + 或 - -)，转换非常简单：byte 类型的操作数或 short 类型的操作数被转换成 int 类型，而其它的类型保持不变。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

对于二元运算符，情况稍微复杂一些。对于只包含整数类型的操作数，如果其中一个操作数是 long 类型，那么其它的操作数也要转换成 long 类型；否则所有的操作数都被转换成 int 类型。表达式的结果是 int 类型，除非是产生的值非常大才使用 long 类型。对于至少有一个是浮点类型操作数的情况，如果表达式中有一个操作数是 double 类型的，那么另一个操作数需要转换为 double 类型，并且计算结果也是 double 类型的；否则两个操作数都被转换成 float 类型，并且结果也是 float 类型的。请考虑一下程序清单 9.1 中的表达式。

幸运的是，隐式转换发生在不需要用户指定或者是不需要用户知道的情况下。编译器处理了所有的与 byte 类型和 int 类型相加有关的细节，所以用户不必知道。

程序清单 9.1　一组含有隐式转换的混合表达式

```
short Width;
long Length, Area;
double TotalCost, CostPerFoot;

// In the multiplication below, Width will be converted to a
// long, and the result of the calculation will be a long.
Area = Length * Width;

// In the division below, Area will be converted to a double,
// and the result of the calculation will be a double.
CostPerFoot = TotalCost / Area ;
```

9.6.2　强制运算符

对于正常地隐式转换，转换是非常自然的，是用户注意不到的。有些时候，确保两种类型的转换是非常重要的。这种类型转换要求显式强制，这就需要使用强制运算符。

强制运算符是由括在括号里的类型名构成的。它是放在操作数前面具有较高优先级的一元运算符，其结果是得到具有由强制指定的类型的变量，并具有原来对象的值。下面是一个显式强制的例子：

```
float x = 2.0 ;
float y = 1.7 ;
x = (int) (x/y) * y;
```

在本例中，当 x 被 y 除时，结果的类型是浮点数。经过强制运算符(int)转换后成为 int 类型，它的结果是 1，不是 1.2。所以最后结果是 x 等 1.7。

并非所有的转换都是合法的。例如布尔型值就不能被强制成任何其它类型，对象也不能被转换成父类。

可参考第 6 章的“声明类”部分。

> **注意**：因为强制包含了无条件类型转换(如果条件是合法的)，所以有时也叫做类型强制。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

9.6.3 强制并转换整数

四种整数类型可以被强制成除布尔型以外的任意其它的类型。强制成较小的类型可能导致数据损失,强制成浮点数(float 或 double)可能导致精度降低,除非整数是 2 的幂(例如 1、2、4、8 等号)

9.6.4 强制并转换字符

字符可以按照 16 位(short)整数执行强制的办法进行强制。也就是说用户可以将字符强制成任何值(除布尔值以外,译者注)。但是如果强制成较小的类型,可能会损失某些数据。

> **注意**:如果用户使用汉字字符集(中文、日文或韩文),那么在将 char 型转换为 short 型(16 位整数)时可能损坏数据,因为最高位将丢失。

9.6.5 强制并转换布尔值

没有任何直接的方法可以强制或转换布尔值为其它类型。如果用户打算使某个整数变量根据当前的布尔值得到 0 或 1 值,可以使用 if 语句实现,或者模仿下面的代码来实现。

```
int j;
boolean tf ;
.....
j = tf ? 1:0 ; //如果 tf 是 true , j 等于 1,否则 j 等于 0
```

9.7 字符串加法

在本章结束之前,让我们讨论一下加法运算符在字符串中的使用。从一般意义上讲,Java 不支持运算符覆盖;但是在字符串联接上是支持覆盖的,Java 是使用"+"运算符来实现字符串联接的。在字符串之间的"+"运算符与在C++里的用法是一样。第一个字符串和第二个字符串相联接产生包含两个串值的新串。在下面的表达式里产生的字符串是"Hello world"

"Hello" + " world"

如果将非串的值加入到字符串中,在联接之前先将非串的值转换成字符串。这就意味着,数字也可以被加到串中。数字值首先被转换成适当的数字字符序列,然后再联接到原来的字符串中。下面的所有表达式都是合法的。

```
"George" + "Burns"
"Burns" + "and" + "Allen"
"Fahrenheit" + 451
"Answer is" + true
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第10章 控 制 流

本章内容：

10.1 控制执行
10.2 布尔型运算的 true 运算符和 false 运算符
10.3 逻辑表达式
10.4 条件运算符
10.5 控制流中的布尔类型
10.6 控制流语句
10.7 循环语句
10.8 跳转语句

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

10.1 控制执行

控制执行流对任何语言来讲都是最重要的一个方面。控制流允许用户为计算机根据条件指示不同的方向。因此,如果用户迷失了方向,就可能随机地选择方向。否则用户将按流程图指示的方向执行程序。

用户也可以将控制流看作是一组红绿灯。当红灯亮时,必须停车,当绿灯亮时,可以继续运行。没有这种决策机制的程序将是臃肿庞大的和没有生命力的。本章将介绍如何使计算机按照"交通图"和"交通规则"运行。

10.2 布尔运算的 true 运算符和 false 运算符

在 Java 中几乎所有的控制流表达式都是基于 true 和 false 值来控制流的。在本章的后面将会介绍:当 if(value)语句中的值(value)为 true 时才能调用执行 if 语句后边的语句。用户可以实际地写一个类似 if(true)语句,但是括号里的内容不是值,而是一个布尔表达式。

用在布尔表达式中的 Java 运算符有着特殊的含义。许多同样的运算符在其它表达式中也使用。在大部分情况下,运算符的意思是由执行整数类型运算引伸出来的。表 10.1 中显示的运算都可以执行布尔运算。

表 10.1 布尔表达式运算

运算	名 字	描 述
=	赋值	就像在 tf = true 中的含义一样
= =	等于	如果两个布尔操作数具有相同的值(同为 true 或同为 false),这种运算的结果为 true。否则结果为 false。这与非异或等价(NXOR)。
! =	不等于	如果两个布尔操作数具有不同的值(一个为 true,另一个为 false),这种运算的结果为 true。否则结果为 false。这与异或(XOR)等价。
!	逻辑非	如果操作数是 false,运算结果是 true,反之亦然。
&	与	只有当两个操作数都为 false 时产生 false 值。注意它只适用于布尔型操作数。对于别的值。它是位运算符。
|	或	只有当两个操作数都为 true 时产生 true 值。注意它只适用于布尔型操作数。对于别的值。它是位运算符。
^	XOR	当只有一个操作数为 true 时产生 true 值。注意它只适用于布尔型操作数,对于别的值它是位运算符。
&&	逻辑与	对于布尔型操作数的作用与 & 相同
||	逻辑或	对于布尔型操作数的作用与 | 相同
?:	If-then-else	问号前面要求布尔表达式

10.2.1 关系运算符

最直观的比较运算符是关系运算符。关系运算符包括标准的大于号和小于号,这是在小学三年级就介绍了的。它的用法与算术中的相应符号用法是一致的。例如,如果写(3 > 4),那么就写错了(false)。另外(3 < 4)是正确的(true)。在 Java 和其它语言中,操作数不仅仅限于常量,也可以使用变量,所以(Democrats > Republicans)也是有效的。下面列出了所有的关系

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

运算符：

运算符	布尔结果
<	小于
< =	小于或等于
>	大于
> =	大于或等于

关系运算符的优先级低于算术运算符，但高于赋值运算符。下面的两个赋值语句产生相同的结果：

```
result1 = a+b<c* d
result2 = (a+b)<(c* d)
```

关联是从左到右的，但这一特性对于关系运算并不适用。请考虑一下下面的表达式：

```
a < b < c
```

第一个表达式 a < b 首先被计算，并产生一个 true 值或 false 值。然后将它与 C 比较。因为布尔值不能用作操作数，所以编译器将产生语句错误。

> **注意**：在 C 和C++ 中，关系运算符产生整数值 0 或 1，这个值可用在任意使用整数作操作数的表达式中。在 C 语言或C++语言中像下面的表达式是合法的：
>
> ```
> RateArray[day1 < day2]
> NewValue = OldValue + (NewRate > OldRate) * Interest ;
> ```

现在编一个简单的程序试验一下前面介绍的关系运算符。程序清单 10.1 的结果告诉用户在三年级的时候学过的一些东西是否正确。这里使用了 Java 提供的另一个方便用法——可以将布尔值加进字符串，并在打印结果中显示 true 或 false。

```
public class QuickTest
{
    public static void main(String args[]){
        System.out.println(" * 5 is greater than 6:" + (5 > 6));
        System.out.println(" * 6 is greater than or equal to 3:" + (6 > = 3));
        System.out.println(" * 8 is less than 10:" + (8 < 10));
    }
}
```

为了运行这个程序，首先将程序清单中的内容拷贝到名为 QuickTest.java 的文件中（文件名中字符大写也很重要，它可以帮助用户记住文件的内容和用途），然后使用 javac 编译该程序：

```
javac QuickTest .java
```

文件编译后就可以运行了：

```
java QuickTest
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用户可能已经猜出了输出结果,如下所示:

```
5  is greater than 6 :false
5  is greater than or equal to 3 : true
8  is less than 10 : true
```

10.2.2 等于运算符

等于运算符是 Java 的另一个比较运算符集。等于运算符允许用户比较两个值并求出它们是否相等。在三年级时,用户可能写过(3 = 3)。遗憾的是,在 Java 中这个语句使得 Java 编译器认为是赋值运算符,而不是等于运算符。在传统计算中,当编译器阅读程序源代码时 = 运算符被解释为 gets(得到),也就是说(3 = 3)的意思是从 3 中得到 3。这并不是用户想要的结果。为了解决这个问题,需要在等于关系中使用双字符的运算符(= =)。这样在 Java 中,用户就可以书写表达式为(3 = = 3),它的意思是 3 等于 3。而等于表达式(3 = = 4)肯定是错误的(false)。

下面的等于运算符非常类似于关系运算符,但优先级稍低一点:

运算符	布尔结果
= =	等于
! =	不等于

等于运算符可以使用任意类型的操作数。基本数据类型的值可以比较。如果操作数同是其它的对象类型(比如用户创建的类),那么计算结果取决于操作数是否引用相同的对象。考虑一下下面的例子:

```
String1 = = String2
```

在这个例子中,String1 和 String2 必须引用同一个字符串而不是两个不同的字符串,即使是字符串中包含相同的字符序列。两个字符串也不相等。考虑一下程序清单 10.2 中的代码行。

程序清单 10.2 ObjectEquals.java——在 Java 中比较对象

```
public class ObjectEquals
{
    public static void main(String args[]){
        String String1 = new String("Hi Mom");
        String String2 = new String("Hi Mom");
        //At this point String1 is not equal to String2
        System.out.println("String1 = = String2 :" + (String1 = = String2));
        String String3 = String1;
        //Now String1 is equal to String2
        System.out.println("String1 = = String3 :" + (String1 = = String3));
    }
```

在程序给出的语句中 String1 = = String2 表达式将返回 false 值,因为尽管它们都包含有

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

相同的字符,但它们不是同一个对象。而另一个表达式 string1 = = string3 将返回 true,因为它们引用的是同一个对象。现在用户可能已经猜出了程序清单 10.2 的输出结果了。如下所示:

String1 = = String2:false

String1 = = String3: true

注意:如果用户需要比较本例中的 String1 和 String2,用户可以使用 String 类的 equals 方法。可以写成 string1.equals(string2)。equals() 方法逐字符比较两个字符串。

这两个运算符的关联性也是从左到右的。用户已经在前面看到了,对于程序员来说,关系运算符的关联性是没有用的。而等于运算符的关联性稍微有一点用处。考虑一下下面的例子:

StartTemp = = EndTemp = = LastRace

在这里变量 StartTemp 和 EndTemp 首先比较,然后比较出来的布尔结果再与 LastRace 比较,LastRace 必须是布尔型的。如果 LastRace 是非布尔型的量,那么编译器将产生一个错误。

警告:编写代码不要太巧妙,不要怕详细繁琐。因为即使用户在编写代码时自己对所编代码十分清楚。但是当编写后几个星期再来读自己的程序可能就和别人读这些程序一样有着神秘感。要尽量使代码易读。如果出于某种原因必须使用像前面例子的表达式,最好使用注释来解释表达式的运算方式。如果可能的话,把编写这种算法的原因也写出来。

10.3　逻辑表达式

第三个比较运算符集是逻辑表达式。逻辑表达式的工作方式与前面介绍的运算有点不同,这可能是小学三年级算术课里没有的内容。

逻辑表达式运算的或者是一对布尔量或者是某个对象的每个位。基于这一点,逻辑运算符可以概略地分成两类:

- 布尔运算符　　只对布尔值进行运算
- 位运算符　　对两个整型操作数的每个位进行运算

用户已经在第 9 章"使用表达式"中看到了位运算符是如何工作的。本章只介绍逻辑表达式运算符的条件部分。除极少的例外外,如果操作数是布尔型的,那么位运算符和条件运算符将产生相同的结果。

10.3.1　条件与运算符和条件或运算符

有两个基本的布尔运算符:

- 条件与运算符:&&
- 条件或运算符:||

奇怪的是在大部分计算机语言(包括 Java)中都没有条件异或运算符。

这些运算符都遵循同一个真值表,在第 9 章"使用表达式"中介绍过位运算符真值表。真值表是非常容易理解的。比如 true && true ,"true 与 true"明显地为 true。为了方便起见,在

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这里再一次列出“与”和“非”的真值表：

当 A 为	当 B 为	(A&&B)	(A‖B)
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

条件或和条件与表达式里的操作数是按照从左到右的次序计算的；如果表达式的值是在左操作数被计算出来之后计算出来的，那么右操作数就不被计算。所以在下面的例子中如果 x 确实小于 y，那么 m 和 n 就不再做比较运算了：

(x < y) | (m > n)

这两个条件运算符的优先级别都低于位运算符。

10.3.2 一元逻辑运算符

Java 有两个一元运算符：

- 对布尔型操作数取反!。
- 对整数类型和布尔型操作数按位取反 ~。

> **注意**：对于整型操作数，这种运算符是按位切换的——操作数中的每一位都被切换。(0 变为 1；1 变为 0)。

将取反操作符放在任何值的前面，都会得到一个与该值本身相反的值，如：! true 的值为 false。

这两个操作数都有较高的优先级，它的优先级与其它的一元运算符的优先级相等。看一下下面的例子，它将逻辑取反和条件与结合起来了：

if (! dbase.EOF & & dbase.RecordIsValid ())

因为逻辑取反有较高的优先级，所以它先进行计算。如果 EOF 指的是文件结尾，那么就需要首先检查是否已经到了数据库文件的结尾。如果没有则继续计算第二个操作数，第二个操作数是一个方法调用，用来判断当前记录是否有效。理解这个表达式的关键是要了解第一个操作数是否为 false，如果不为 false，也就是说已经到了文件的结尾，那么就不必再检查记录是否可用了。

10.4 条件运算符

条件运算符是唯一的一个三元运算符，它有三个表达式操作数(不是典型的两个操作数)，在 Java 中的用法和在 C 和C++ 中的用法是一致的，都使用下面的格式：

表达式 1 ? 表达式 2 : 表达式 3

在这个语句中，表达式 1 必须产生一个布尔值。如果表达式 1 的值为 true，那么表达式 2 被计算。如果表达式值为 false，那么表达式 3 被计算，并且整个表达式的值是有条件的。三元

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

操作符与典型的 if 语句十分相似：

```
if (表达式 1)
表达式 2 ;
else
表达式 3 ;
```

考虑一下下面的例子。第一个语句使用条件运算符求出两个值的最大值；第二个语句求出了两个值的最小值；第三个语句求出了两个值的差的绝对值

```
BestReturn = Stocks > Bonds ? Stocks : Bonds ;
LowSales = JuneSales < JulySales ? JuneSales :JulySales ;
Distance = Sitel - Site2 > 0 ? Sitel - Site 2 : site2 - sitel ;
```

上面的例子，都使用了优先级规则。仅使用优先级规则不使用括号也可以得到正确的结果。

10.5　控制流中的布尔类型

布尔类型(和布尔表达式)是唯一可用在控制流语句中的数据类型。如下面程序片段所示：

```
Bolean TestVal = false ;
int IntVal = 1
...
if (TestVal ){}else {}
if (IntVal ! = 1){}else{}
....
While (TestVal ){}
While (IntVal = = 0){}
.....
do {}while(Testval)
do {}while (IntVal = = 0)
for (int j = 0 ;TestVal ;j++ ){}
for (int j = 0 ;IntVal < 5 ;j++ ){}
```

在这个代码片段中，整数变量 IntVal 与常值比较是非常简单的布尔表达式。当然也可以在这个位置使用更复杂的表达式。

10.6　控制流语句

控制流是程序的大脑。控制流能够调整程序进程流的方向，通过调整方向程序变得生动灵活。没有控制流，程序除了执行几个顺序的操作外就什么也做不成了。

10.6.1　if 语句

最简单的控制流形式是 if 语句。if 语句有一个条件表达式(可以有多种形式)，如果该表

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

达式的值为 true,则执行下一个代码块。if 语句的语法是:

if(表达式)
语句

如果表达式的值为 false,计算机跳过该语句继续执行。下面的代码片段是一个 if 语句的示例:

```
if(MyNameIsFred)
System.out.prinfln("Hi Fred")
System.out.println("Welcome to the system");
```

如果 MyNameIsFred 的值为 true,当运行这个程序片段时,计算机将打印下列内容:

```
Hi Fred
Welcome to the system
```

如果 MyNameIsFred 的值为 false,那么程序就跳过 if 语句的下一行。此时打印结果是:

```
Welcome to the system
```

在多数情况下,用户将希望在条件成立时执行多行代码。为了做到这一点,用户可以在 if 语句后面放上一个代码块,代码块的开始处和结束处放一对花扩号。下面的代码片段就是这样一个示例:

```
if (umpire.says.equals("Strike two")){  //equals 方法返回布尔型值
Crowd.Cry("Fraud");                     //方法调用
Strike++;                               // if 块的最后一条语句
}
Casey.face("Christian charity");        // if 块后面的第一个语句
```

10.6.2 if-else 语句

仅仅比简单的 if 语句进了一步,if-else 语句在条件表达式的值为 false 时执行 else 语句。如果 if 表达式的值为 true,那么 else 块的代码就不执行了。if-else 语句的语法是:

```
if(表达式)
        if 语句;
else
        else 语句;
```

下面是一个 if-else 语句的示例:

```
if(Strike! = 2)
Casey.lip("Curling Sneer");          //单语句(可以是语句块)
else{
Casey.teeth("Clenched in hate");     //语句块(可以是单语句)
Casey.bat.pound ("Plate");
}
```

if-else 语句一个很重要的方面是:当有嵌套的 if 语句时,else 块怎样执行?

考虑一下下面的代码:

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
if(firstVal = = 0)
if(secondVal = = 1)
firstVal + + ;
else
firstVal - - ;
```

什么时候执行 else 语句？在本例中 else 是在第二个(内嵌)if 语句中的。if - else 语句被当做了一个语句。所以 else 属于最近的语句。另一种使程序按照先进先出(FIFO)的风格组织的结果是 else 属于第一个 if 语句。把第二个 if 语句放在一个块中就可以做到。

```
if(firstVal = = 0){
if(secondVal = = 1)
firstVal + + ;
}
else
firstVal - - ;
```

因为块是被作为单语句来处理的，所以 else 与第一个 if 相关联。

还有一种合法的 if - else 语句，叫做复合 if 语句。

```
if(firstVal = = 0)
  if(secondVal = = 1)
    firstVal + + ;
  else if(thirdVal = = 2)
firstVal - - ;
```

在这个例子中只有在 firstVal 为 0，secondVal 不为 1 和 thirdVal 为 2 时 firstVal - - 语句才能够得到执行。

10.7　循环语句

程序员使用循环语句根据循环条件控制重复执行语句序列。

Java 支持五种循环语句：

- While
- for
- break
- do
- continue

除了 Java 中 continue 语句和 break 语句可以带能够改变它们的行为的参数外，这些语句的用法与 C 和C++ 中的语句的用法是一样的。(在 C 和C++ 中这些语句都不带参数)。

10.7.1　while 语句

while 语句测试表达式的值，如果值为 true，则重复执行下一个语句或语句块直到表达式的值变为 false。当变量或表达式为 false 时，控制就跳过 while 语句的下一个语句。下面是 while 循环的语法，它与 if 语句有些相似：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

while(表达式)
语句;

语句可以有意地或无意地变成死循环,只要表达式的值无法变为 false 就会出现这种现象。下面的示例就可能是一个死循环:

```
while(Casey.RoundingTheBasepads = = true){
Crow.cry("Hooray for Casey");
```

在这个示例中,表达式的初值可能不为 true 如果不为 true,那么子语句块就永远得不到运行。如果它为 true,这个代码块就一直重复执行,直到表达式的值不为 true(因为块中,没有改变 Casey.RoundingTheBasepads 的值的语句,那么表达式的值也一直得不到改变,所以就成了死循环。译者注)。

10.7.2 do 语句

do 语句与 while 语句很相似。事实上 do 语句有一个 while 从句在尾部。while 语句的表达式必须是布尔型的。do 循环执行完语句后计算 while 表达式。如果 while 表达式为 true,重新执行 do 语句,直到表达式的值变为 false。

do - while 循环的完整语法是:

```
  do
  语句;
while(表达式)
```

程序员选用 do 语句代替 while 语句的主要原因是 do 语句至少执行一次,不管表达式的值为什么。这也叫做后验证。下面是 do 语句的示例:

```
do{
Crowd .cry("Kill the Umpire!");
}while(umpire.say.equals("Strike two"));
```

在这个例子里,不管别的条件如何,Crowd.cry 至少被调用一次。只要 umpire.says 方法返回字符串"Strike two",Crowd.cry 方法就一次又一次地被调用。

10.7.3 for 语句

for 语句是四种循环语句中最复杂的一种。程序员可以用它代替上述三种语句。for 循环的完整语法是:

```
for(初始化;表达式;步长)
语句;
```

for 循环首先运行初始化代码(与 do 循环类似),然后计算表达式(类似于 if 语句和 while 语句)。如果表达式的值为 true,语句被执行,然后执行步长。for 循环也可以写成 while 循环:

```
初始化
while(表达式){
语句;
步长;
}
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

下面的程序片段中包含有 for 循环：

```
for(int ball = 0,int strike = 0;(ball < 4) & &(strike < 3); Ump.EvaluateSwing()) {
Pitcher.pitch();
Player.swing();
}
```

在这个例子中，初始化部分有多个语句，语句之间是用逗号分隔开的。初始化语句和步长语句可以有多个语句。另外这两个语句可以为空，即可以没有这两个语句。

10.7.4　switch 语句

接下来的控制流的方式是 switch 语句。switch 语句是第一个不要求布尔计算的控制流语句。switch 根据语句中的表达式的值来传递控制给多个子语句中的一个。switch 将控制传给语句标志 case 后面的值与表达式的值相同的第一个语句。否则 switch 将控制传给含有 default 标志的语句。如果没有 default 标志，控制将被传给 switch 块后面的第一个语句。

switch 语句的语法是：

```
switch(表达式){
  case   值 1:   语句 1;
  break;
  case   值 2:   语句 2;
  break;
  default:       语句 D;
  }
```

Switch 语句的特别之处是：它的表达式必须是整数类型的。用户可以使用 byte 型，shorts 型、char 型或 int 型，但不能是 float 型和 Boodean 型。

break 语句不是必须的。因为 switch 工作方式的原故，经常使用 break 来结束 switch 的执行。只要值与表达式匹配，程序就从这个点开始一直执行下去。所以后面的语句全部会被执行。现在看一下下面的示例。

```
switch(1){
case 1:System.out.println("one");
case 2:System.out .println("two");
default:System.out .println("Default");
}
```

在这个示例中，输出结果如下所示：

```
one
two
Default
```

出现这个结果是因为第一个输出语句的条件值(1)与表达式的值(1)相匹配，所以 switch 语句块从第一个输出语句开始执行，直到结束。很有可能用户并不希望打印所有三个结果。使用 break 语句可以只产生一个打印输出。为了做到这一点，上述代码应当改成下面这样：

```
switch(1){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
case 1:System.out.println("one")
break;
case 2:System.out.println("two");
break;
default;System.out.println("Default");
}
```

注意:与 if 语句、while 语句、do 语句和 for 语句不同,case 语句不受单语句限制。也没有块要求,从匹配的条件句开始执行直到遇到 break。

switch 表达式的值和 case 标记常量必须是整数类型的,可以是 byte 型、short 型、char 型或 int 型。另外,在同一个 switch 块中不能有两个 case 标志使用相同的值。

下面的代码片段是包含了另外一个 switch 语句的示例

```
switch (strike){
case 0:
case 1:
Casey.lip("Curling Sneer");
break;
case 2:
Casey.teeth("Cleached in hate");
Casey.bat.pound("Plate");
break;
default:
System.out.println("Strike out of range");
}
```

在这个示例中,假设 strike 是一个兼容的整数类型(例如是 int)。控制传递方向取决于 strike 的值。如果 strike 的值与应该使用的所有值都不同,那么 switch 应该打印一则程序员定义的错误信息。

10.8 跳转语句

Java 有三种跳转语句:break 语句,continue 语句和 return 语句。

10.8.1 break 语句

使用 break 语句终止循环的子语句块和 switch 语句。无标志的 break 语句将控制传递给当前循环(while 语句,do 语句,for 语句和 switch 语句)的下一行。

如果有标志,控制会被传递给当前方法中带有这一标志的语句。如果一个开放的 try 语句有一个 finally 从句,则从句将在控制被传递之前执行。

10.8.2 continue 语句

continue 语句只出现在循环语句(while 语句、do 语句或 for 语句)的子语句块中。无标志

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

的 continue 的作用是跳过最里层循环语句块的剩余部分直接执行下一个循环。标号参数允许程序员选择要继续执行的循环层。

如果在指定的嵌套层中当前开放的 try 语句有一个 finally 从句。则从句会在控制传递之前被执行。

10.8.3　return 语句

return 语句将控制传递给方法、构造器或含有 return 语句的静态初始化器的调用者。如果 return 语句位于没有 void 声明的方法中，那么它可能拥有与方法相同的类型。

如果一个开放的 try 语句有一个 finally 从句，则从句会在控制传递之前被执行。

参见第 8 章的“返回信息”部分。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 11 章　类

本章内容：

11.1　什么是类
11.2　为什么使用类
11.3　Java 中的类
11.4　声明类
11.5　构造器
11.6　创建类的实例
11.7　引用类的部分
11.8　变量
11.9　嵌入类
11.10　包
11.11　引入包中的类
11.12　引入整个包
11.13　不通过引入使用类
11.14　使用包来组织代码
11.15　所有 java.lang 类的隐式引入

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

11.1　什么是类

类是面向对象结构的主要构造块。没有类也就没有对象，面向对象编程也就成了面向程序的编程，那样就没什么意思了。使用对象有几个优点。类允许封装数据，将与某个项目有关的信息和动作与其它的代码分别存放。类允许继承，有了继承，用户就可以从一个简单的类开始建立越来越复杂的类。通过多态技术，共有某一属性的不同对象可以共享他们的相似性。

从一般意义上看，类是汇集一套数据并定义所有访问、使用和修改这些数据的方法。

基本上每个类都有两个部分。第一部分是状态。对象的状态是类中所有变量的值。例如，定义一个有一个变量 RedGreenYellow 的类 StopLight，那么 StopLight 的状态是由 RedGreenYellow 的值决定的。如下所示：

```
Public class StopLight {
        int RedGreenBlue
}
```

类的第二部分是工具或方法。类的方法决定了类有哪些可利用的手段。在 StopLight 例子中，可能有一个叫 ChangeLight ( )的方法，它使红绿灯由红变绿(通过 RedGreenYellow 变量)。

```
Public class StopLight {
        int RedGreenBlue ;
        changeLight ( ){
                RedGreenBlue = ++ RedGreenBlue%3;
        }
}
```

注意：为了区分类的变量和方法中的成员变量，类的变量通常又叫做域，或类范围变量。在上面的例子中，RedGreenBlue 变量是 StopLight 的域。

11.2　为什么使用类

当用户与类打交道时，记住类并不能够使用户做比没有类时更多的事情，有了类可以编写面向对象的程序，从而使程序员工作效率提高。

为什么要使用类？这个问题的答案与大公司分成几个部门和若干个子部门的原因是一样的。一般公司通过把有数千个任务的数百人分成若干个部门，各部门负责分配任务和责任。因为财务部知道如何为顾客开票，销售部不需要关心有关票据具体的要求，所以财务部门将票据封装在他们自己部门之内。

面向对象编程不仅仅是具有封装能力，大量的 OOP 都提供了继承能力——基于已有类建立新类。例如用户想编一个国际象棋游戏，并且以前编过西洋跳棋程序。如果是使用传统的编程技术，可能需要从零开始编起或者从以前编过的代码中裁剪部分内容。但如果使用继承则可以减少大量工作。继承西洋跳棋类，只要覆盖西洋跳棋类中的那些与国际象棋不同的方

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

法和增加国际象棋中特有的方法即可。

注意:当新类继承其它类的属性时,新类就可被称为子类或孩子类。而被继承的类叫父类或超类。

封装是 OOP 的一大特征,即将数据和方法封装起来使得这方面的信息与程序的其余部分有效地分离开。一旦用户创建了一个完整的、能够执行一定任务的类,就可以不管该任务的细节而直接调用由该类提供的方法。因为类机制是独立的,即使用户后来严重地修改了类的内部工作,只要对方法的调用没有改变就不必修改程序的其余部分。类封装的另一个好处是通过在类中放置数据和创建适当的方法来操作这些数据,用户可将程序中剩下的数据封装起来,这样可以防止意外的数据损坏。

多态性可以说是 OOP 最具诱惑力的特性。类创建的自维护组件可以被该类的子类拓展和提高性能,而此时的子类和它的父类被认为是同"态(型)"的。OOP 的这一特性就叫做多态性,它允许用户对不同类型的类执行同一个操作,只要这些类具有某个共同的特性即可。虽然每个类的行为可能不同,但仍可对它们执行同一个操作,因为它们都在同一棵氏族树上。例如,如果要创建一个 Vehicle 类,稍后还要创建 Truck 类和 Bike 类,这两个类都可以在 Vehicle 类的基础上扩展出来。虽然摩托车(bike)和卡车差别很大,但们都是机动车(vehicle)。所以 Vehicle 类的实例能做的事情,Truck 类和 Bike 类的实例也能做。对于汽车销售商来说,不管是大富豪(Volvo)汽车还是萨顿(Saturn)汽车都可以卖钱获利。

面向对象编程的新思想

OOP 强调制成标准组件的编程思想。它要求用户将任务分成可管理的部件,并使每个部件具有一定的功能。过程语言中的函数只是简单地拼凑程序,而对象是活生生的"家伙"(creatures),它具有自我管理能力,能够与其它操作共同运行,甚至在程序的其它部分都已终止的情况下仍然存在。正是对象的这种独立生存能力和将对象作为独立实体来工作才使得 OOP 十分地适合 Java 这种基于网络的语言。

警告:在前面的例子中,虽然摩托与和卡车都是机动车,但机动车不一定是摩托车和卡车。所以在 Java 中 Bike 类和 Truck 类可以被看作是 Vehicle 类。但是不能在 Vehicle 类的实例中执行 Bike 类的扩展操作。

11.3 Java 中的类

正如本章开始所陈述的那样,类是 Java 小应用程序和应用程序的基本构造块。类的用途是创建对象。当用户创建某个类的实例时,实际上是创建了一个对象。用户可以为对象包括类中的所有代码。根据面向对象的示例,用户可以使用类将国际象棋的游戏重新创建或改进现有的程序。

更大更好的 Java

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

Java 本身就是使用公用的 JDK 编写的。尽管还存在一些局限性，但大多数 Java 类可自我扩展。为了做到这一点，用户可定制 Java API 库，特别是 AWT 中的类来满足用户的特殊需求。

在开始创建大程序之前，用户必须学会如何创建简单的类。从语法上讲，Java 的类可以分为两部分：声明和体。程序清单 11.1 中有一个简单的类，该类用于创建前面讲的国际象棋游戏的棋盘。仔细考虑该清单总结出建造类的思想。用户可以随着对 Java 的进一步理解，以后可以再来查阅这个程序清单。

```
public class GameBoard
{
/* This is the beginning of a simple game board class that provides the basic */
/* structures necessary for a game board.  It may easily be */
/* extended to create a richer game board. */

    private static final int WIDTH =      10; /* These are constants */
    private static final int HEIGHT = 10; /* that you want to */
    private static final int EMPTY = 0; /* keep as standards */

    private int board[][];
        // This array will keep track of the board

    public String myname;                    // what game is being played

    public GameBoard (String gamename) {
        board = new int[WIDTH][HEIGHT];
        myname = new String(gamename);
    }

    public final void cleanBoard() {
        for (int i = 0; i < WIDTH; i++)
            for (int j = 0; j < HEIGHT; j++)
            board[i][j] = EMPTY;
    }

    public synchronized void setSquare(int x, int y, int value) {
        board[x][y] = value;
    }

    public synchronized boolean isEmpty(int x, int y) {
        if (board[x][y] == EMPTY)
            return(true);
        return(false);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

浏览一下这个类可以看到:任何类的第一部分都是类的声明。用户以后将要写的类大部分也都与 GameBoard 相类似:

Public Class GameBoard

类声明中讲了几个东西,但最重要的可能就是类的名字(GameBoard),对于任何公共类,类名必须与存放它的文件的名字匹配。换句话说,文件名中必须有类名出现。如存放 GameBoard 类的文件名为 GameBoard.java。

类中接下来的部分是开始花括号。用户应该注意到了在类的开始处有一个花括号({),如果看一下程序的底部就会发现还有一个结束花括号(})。这一对花括号定义了文件中存放类定义的区域。

接下来用户将看到几行注释。正如在"注释"(第 7 章)部分所介绍的那样:注释可以放在文件的任何位置并可以被编译器忽略掉,它的作用是为程序员自己或其它程序员存放程序或与本语句有关的解释信息。接下来是几个域声明。这里的每个变量都可以被本类中的方法访问。当用户在某个方法中修改它们时,所有的其它方法都可以看到新值。

```
private static final int WIDTH = 10; /* These are constants */
private static final int HEIGHT = 10; /* that you want to */
private static final int EMPTY = 0; /* keep as standards */
private int board[][];
// This array will keep track of the board

public String myname;              // what game is being played
```

Finally, you should see four methods:

```
public GameBoard (String gamename) {
    board = new int[WIDTH][HEIGHT];
    myname = new String(gamename);
}

public final void cleanBoard() {
    for (int i = 0; i < WIDTH; i++)
        for (int j = 0; j < HEIGHT; j++)
            board[i][j] = EMPTY;
}

public synchronized void setSquare(int x, int y, int value) {
    board[x][y] = value;
}

public synchronized boolean isEmpty(int x, int y) {
    if (board[x][y] == EMPTY)
        return(true);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        return(false);
    }
}
```

11.4　声明类

一般情况下 Java 类的声明使用如下格式：

访问说明符 class *新类 extends 超类名 implements 接口名*

在这里斜体内容是可选的。从上面的格式可以看出，在类声明中可以为类定义四个属性：

- 修饰符
- 类名
- 超类名
- 接口

11.4.1　访问说明符

声明中的访问说明符确定了在以后开发中如何处理该类，它与在第 8 章“方法”中讨论的四种访问说明符很类似。虽然在开发类本身时它并不重要，但在决定包含该类去创建其它的类、接口和异常时它就变得非常重要了。

在创建类时，用户可以接受缺省值，也可以使用三种说明符：public、final 和 abstract 中的一个。

11.4.1.1　public 类

通过将修饰符 public 放在类声明之前来定义该类为 public(公共类)。就像公共类的名字暗示的那样，公共类可以被所有对象访问。这意味着它可以被任何对象使用和扩展，与它的包无关。例如：

```
public    class    PictureFrame
```

还应该注意：公共类必须定义在名为“类名.Java”的文件中(例如 PictureFrame.java)。

11.4.1.2　protected 类

如果在类声明之前没有放置任何修饰符，那么类就使用 protected 这种缺省的属性来创建。有些用户应该已经意识到了这个属性的含义。

在缺省情况下，所有的类均被赋于 protected 访问级。这意味着虽然该类可以被扩展和被其它的类调用，但是只有那些在同一个包中的对象才可以使用该类。下面是一个 protected 类的例子：

```
class    ProtectedFrame
```

11.4.1.3　final 类

final 类没有任何子类。可以通过将修饰符 final 放在类声明的前面来创建 final 类。

首先，创建 final 类的原因可能不明显。那么为什么要防止使用类来扩展其它类呢？这岂不是与面向对象的宗旨相违背么？

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

记住面向对象编程允许用户为某个类创建许多版本(创建子类只能继承属性,但不能修改属性)是很重要的。但是如果用户正在创建一个用作标准的类(比如处理网络通信的类),用户当然不希望允许其它的类用不同的方式操纵这个功能。使用 final 类用户可以消除这种修改的可能性,并可以保证程序的一致性。

另外,告诉编译器这是某个类的最终版本,可以允许编译器执行运行代码优化。其它类型的类没有这种能力。下面是 final 类声明的示例:

final class PictureFrame

11.4.1.4 abstract 类

抽象类由修饰符 abstract 指示。抽象类至少有一个方法是不完整的。"抽象"指的就是这种不完整。下面是 abstruct 类声明的示例

abstract class PictureFrame

一个已编写完的类怎么能是不完整的呢? 许多语言都有语法检查类,它们的方法是基于具体的语言版本中的类来实现的。为了建立一个清洁器程序,可以简单地编写一个 GrammarChecker 类或 SpanishChecker 类,在 GrammarChecker 中将与语言有关的方法声明为 abstract 型并空着不写内部的语句。然后创建与语言有关的类,这时就可以扩展抽象的 GrammarChecker 类了,使用实际的代码重新定义这些空方法。虽然最后还是需要为各种语言编写单独的类,但是代码的核心是在 GrammarChecker 类中,只是依赖具体语言的那部分是空着的。

注意:因为类不完整,所以不能创建抽象类的实例。抽象类的声明不必太复杂,可以简单,在上面的例子中只使用了一个修饰符 public,没有使用其它的类和接口。

11.4.2 类名

像所有其它的 Java 标识符一样,对类名也只有下列要求:

- 使用字符或下划线_和货币符 $ 开头。
- 只包含统一码中大于十六进制数 0x00C0 的字符(基本字符和数字以及某些其它的专用字符)
- 不允许使用 Java 关键字(比如 void 和 int)。

类名首字符大写不是必须的,但从实用角度看,它是一个良好的习惯。

提示:虽然只有公共类要求类名与文件名一致,实际上其它访问类型的类这样做也是有好处的。例如将 newclass 类定义在 NewClass.java 文件中,这样做可以帮助编译器寻找 NewClass,甚至在 NewClass 类还没有被编译时也可以帮助程序员记住 NewClass 类。

11.4.3 超类——扩展另一个类

OOP 最重要的一个方面是能够使用已建类的方法和域。通过基于简单的类建立较大的类的办法,用户可以减少许多代码的编写。更重要的是用户可以大大地减少在程序中查找和修改错误的工作。为了在已有类的基础上建立新类,必须在类声明中使用扩展声明。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

通过扩展超类,用户可以得到该类的一个拷贝,并可以在这个拷贝中增加新内容。如果不做其它工作(也不使用修饰符)那么新建类的行为与原来类的行为是一致的。新类将具有与超类完全相同的域和方法。

> **注意**:下面的例子是否熟悉:
>
> public class MyClass extends Applets {
>
> 如果查看任何小应用程序的源代码,都可以看到类似的声明语句。事实上,用户可能是在不知不觉中扩展着 java .applet .Applet 类。
>
> 是否记住用户在小应用程序中使用过的方法,如 ShowStatus( )、init ( )和 keyDown ( )?它们是凭空来的吗? 不,它们是 java.applet.Applet 类或 java.applet.Applet 扩展类如 java.awt.Component 类的方法。通过扩展 java.applet.Applet 类,用户可以在自己的小应用程序类中访问和使用这些方法,这样做可以为小应用程序提供许多功能。

Java 中的每个类都可以被当成对象。在缺省情况下,每个类都是从 java.lang.Object 类派生出来的。所以如果用户类没有扩展任何其它类,但它还是扩展了 java.lang.Object 类。

> **注意**:在 Java 中不存在多继承。这一点与C ++ 不同,Java 的类只能扩展一个类。

11.5　构造器

构造器是具有特殊属性和特殊用途的特殊方法。构造器用于在建立类的实例时设置某些属性和执行某些任务。下面的代码是 GameBoard 类的构造器:

```
Public GameBoard (String gamename ){
    board = new int [WIDTH][HEIGHT] ;
    my name = new String (gamename );
}
```

构造器使用与所属类相同的名字。在 GameBoard 类中,构造器的名字是 GameBoard( )。其次,构造器不指定返回参数,因为它们不是实际调用的方法。例如,如果要创建 GameBoard 类的实例,用户可以使用下列代码:

GameClass myGame = new GameClass ( )

当 new GameClass( )实际创建实例时,调用构造器方法。

一般地,构造器用于初始化类的域并执行与创建有关的任务,比如与服务器的连接和初始化计算。

还要注意:重载构造器时允许用户以不同的方式建立对象;用户可以创建几个构造器,这些构造器拥有不同的参数集。这样就可以创建不同的 GameBoard 实例,可以指定 myname 值、board 值、同时指定 myname 值和 Board 值或者都不指定。在 Java 库中有许多这种做法。通过指定变化程度和信息类型,用户可以建立多种数据类型。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

提示:大部分程序员为构造器选择 public 访问类型。这是因为如果构造器的访问级低于类本身,那么其它类可以声明类的实例,但不能够实际建立该类的实例。

这种漏洞可以为用户提供帮助。通过将构造器声明为私有,可以允许其它类使用该类的静态方法,但不允许建立该类的实例。

应当注意:构造器不能被声明为 native ,abstract , static ,synchronized 或 final 。

11.5.1 覆盖方法

在同一个类中有两个具有相同名字和相同参数表的方法是不合法的。总之,这样做将导致整个系统的混乱(哪个方法是用户想调用的?)。扩展类的目的是创建具有增加功能的新类。为了允许用户这样做,当继承其它类时,用户可以使用与超类相同的名字和参数表重新定义方法。这种行为称为覆盖。例如程序清单 11.2 中的 Elevator 类。

```
class Elevator {
    ...
    private boolean running = true;
    ...
    public void shutDown() {
        running = false;
    }
}
```

用户可能认为这样做不是非常安全的,所以决定创建一个较安全的类。可以在扩展老的 Elevator 类时保留它的大部分属性,只修改一部分。具体地说,用户可能希望证实在电梯停机时厢内是空的,所以就使用程序清单 11.3 中相应的代码覆盖原来的 ShutDown()方法。

```
class SaferElevator extends Elevator {
    ...
    public void shutDown() {
        if ( isEmpty() )
            running = false;
        else
            printErrorMessage();
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意：只有些新的方法与父方法的名字和参数完全相同时才能实现覆盖。如果参数不同，新方法将重载父方法，但不覆盖它。如程序清单 11.4 所示。

```
class SaferElevator extends Elevator {
        ...
        public void shutDown(int delay) {
            if ( isEmpty() )
                running = false;
            else
                printErrorMessage();
            }
    }
```

Elevator 类的 ShutDown 方法并没有发生变化。为方法增加参数(int delay)叫做方法签名。新的 ShutDdown 方法仍然合法，仍可调用，因为它重载了原来的 ShutDown 方法；这和第 8 章“方法”中的做法是一样的。

注意：当用户覆盖某个方法时，用户可以使它比原来的方法更具有安全性。因为 Elevator 的 ShutDown 方法是公共的(public)，所以用户不能在 SaferElevator 中将它指定为私有的(private)。

11.6　创建类的实例

为了实际使用用户创建的类，用户需要创建该类的实例。实例是该类型的类的对象。用户创建的任何类都可以被实例化，就像 Java 中的其它数据类型一样。例如，为了建立 GameBoard 类的实例，用户需要将某个变量声明为 GameBoard 类型

程序清单 11.5 中名为 Checkers 的类创建了一个 GameBoard 类的实例。

```
public class Checkers{
  GameBoard myBoard = new GameBoard();
    ....
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:从技术方面讲,为了使用类而创建类的实例并不是必要的。如果用户调用的方法或变量是静态的,就可以不创建实例。但是静态的方法只能引用静态的类变量。

有一个例外是允许用户访问 System 类的 out 变量时,不需要创建 System 变量的实例,换句话说:用户可以使用下面的代码:

```
System.out.println ("Note,System was not instantiated !!");
```

用户可能已经注意到:声明对象类型,如 GameBoard 类和声明基本类型(如 int m)的方式是不同的,声明类的对象使用的是 new 关键字。在程序清单 11.5 中我们使用了短语 new GameBoard()创建了 GameBoard 类的实例。new 关键字执行下列几个关键性任务:

- 告诉计算机分配存储 GameBoard 对象的空间
- 引起 GameBoard 类的构造器方法的调用
- 为对象返回一个引用(马上赋给 myBoard)

用户可能会对此感到奇怪,类的第一个实例是怎么创建的?在用户创建小应用程序或应用程序时,用户并没有直接为类建立实例的方法,那么实例是怎么来的?答案在于虚拟机。

当浏览器遇到 < APPLET > 标记(要想更多地了解小应用程序,请阅读第 14 章"编写小应用程序")时或当 Java 程序在一个应用程序(见第 17 章"小应用程序和应用程序比较")上运行时,虚拟机要做几件事情。对于应用程序,当用户输入 java MyClass 时,虚拟机调用 MyClass 类中的静态方法 main()。在这个过程中 Java 并没有实际创建 MyClass 的实例,但是因为它是静态的(见前面的注意部分)所以实例不是必要的。这就是需要在 main( )方法中创建类实例的原因。对于小应用程序,当浏览器遇到 < APPLET CODE = "MyClass" > 时浏览器会自动地调用 init()方法。

注意:Java 中对象和基本类型的另一个差别是引用方法不同。基本类型总是直接引用值。对象类型总是被引用。这意味着在下面的代码中最终的 x 和 y 是不相等的,但 w 和 z 的 myName 是一致的:

```
int x = 5
int y = x ;
y ++ ; // x = 5, y = 6
GameBoard w = new GameBoard ( ) ;
GameBoard z = w ;
w. myName = "newString" ; //因为 z 和 w 指向同一个对象,所以它们都具有相同的 myName 值。
```

11.7 引用类的部分

鉴于用户已经开始了开发类的工作,有必要考察一下类如何被其它类使用。正如前面"为什么使用类?"部分介绍的那样,Java 的类可以包含作为变量的其它类的实例。用户也可以处理这些类的域和方法。为了做到这一点,Java 采用了大部分 OOP 语言使用的圆点标记。如程序清单 11.6 所示。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

范例清单 11.4　Checkers.java——GameBoard 对象被 board 所引用

```
import java.awt.*;
    public class Checkers
    {
        private GameBoard board;

        public Checkers() {
            board = new GameBoard("Checkers");
            board.cleanBoard();
        }
        ...

        public void movePiece(int player, int direction) {
            java.awt.Point destination;
            ...
            if (board.isEmpty(destination.x, destination.y) )
             // code to move piece
        }

        private void showBoard(Graphics g)
            g.drawString(board.myname,100,100);
            drawBoard(g);
        }

        private void drawBoard(Graphics g){
        ...
        }
}
```

注意在 GameBoard 类中 board 是一个实例，GameBoard 类的 myname 变量可以被 board.myname 引用。这种访问的一般表达法为

实例名.方法或变量名。

警告：

变量名 myname 是被当作 board.myname 来引用的，并非 GameBoard.myname 如果用户试图这样做，将会得到如下所示的编译错误：

Checkers.java:5: Can't make a static reference to non-static variable myname in class GameBoard

这是因为一个 GameBoard 是类的类型，而 board 是该类的实例。正如前面介绍的那样，当用户处理 board 时，用户实际在处理 GameBoard 类的一个特殊拷贝。因为 myname 不是静态变量，所以它不是 GameBoard 类的属性，但它是该类的实例的属性。所以它不能被改动和通过将 GameBoard 作为变量名来引用。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

11.7.1 特殊变量 this

前面已经介绍了如何引用其它类。但是,如果用户希望类引用类本身该怎么办? 虽然这样做的原因不明显,但是引用自身对于类来说是很重要的能力。为了解决这个问题,当需要引用类自身时可使用特殊的变量 this 。

通常,在下述两种情况下需要使用 this 变量:

- 当在类中有两个变量同名,其中一个属于类而另一个属于某个特定的方法。
- 当某个类需要将类作为参数传递给某个方法。在用户创建使用其它类的小应用程序时,往往希望为类提供像 ShowStatus()这样的方法。例如,如果用户正在创建一个 Presentation 小应用程序,并希望使用简单的 TextScroll 类在屏幕的底部的状态条中显示某些文本,那么就需要为 TextScroll 类提供某些使用该小应用程序的 ShowStatus()方法的手段。达到这一目的最好办法是在创建 TextScroll 类时,使用构造器方法,并将 Presentation 小应用程序类的一个实例作为一个参数。

在下面程序清单 11.7 的例子中,TextScroll 类在 Presentation 类的屏幕底部显示信息。

```
public class Presentation extends Applet {
        TextScroll scroller;

        public void init() {
            ...
            scroller = new TextScroll(this, length-of-text);
            scroller.start();
        }
            ...
    }

    class TextScroll extends Thread {
        Presentation screen;
        String newMessage;
        boolean running;
        int size;

        TextScroll(Presentation appl, int size) {
            screen = appl;
        }

        public void run() {
            while (running) {
            displayText();
            }
        }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    void displayText() {
        // perform some operations to update what should
        // be displayed (newMessage)

        screen.showStatus(newMessage);
    }
```

参见第 13 章的“什么是线程”部分。

注意 Presentation 类的 init()方法的特殊变量 this 的使用方法和使用结果。这一技术是非常有用和强大的。

11.7.2　特殊变量 super

特殊变量 super 提供了对类的超类的访问。当覆盖方法时这个变量很有用，因为这时用户可能希望使用老方法中的代码。例如，如果用户正在创建一个扩展 GameBoad 类的新类 NewGameBoard 并且覆盖 SetSquare()方法，这时用户可能要使用 super 变量来获取原来的代码，从而免去拷贝之烦恼(见程序清单 11.8)。

```
class NewGameBoard extends game board {

    private static int FIXEDWALL = 99;
    // permanent wall, cannot be moved

public static synchronized void setSquare(int x, int y, int value){
    if (board[x][y] != FIXEDWALL) {
        super.setSquare(x,y,val);
    }
}
```

在前面的列子中，用户用 super 变量引用了 GameBoard 类中原来版本的 SetSquare()方法，使用这种办法可以免做令人头疼的方法拷贝工作，同时还可以增加 SetSquare 方法的功能。这种做法允许用户将 SetSquare 的功能封装在原来的 GameBoard 类中，如果用户发现代码中有逻辑错误，也不必将 GameBoard 类和 NewGameBoard 类全部都修改，只需将含有错误的类加以修改即可。

用户应该考察一下，当正要处理的方法是构造器方法时如何调用超(super)方法，调用父类的构造器和调用其它类的构造器一样重要。调用父类构造器与调用其它超(super)方法差别不大，很容易搞混淆：

```
public NewGameBoard (String gamename ){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// 新代码从这里开始
super(gamename );
}
```

这是最简单的写法，super 可以被认为等同于 GameBoard。因为 GameBoard ( )是原构造器方法的名字，所以可以使用 super()形成对原方法构造器的引用。

11.8 变量

很明显，变量是程序和类的一个组成部分。在第 7 章“数据类型及其它标记”中用户考察了各种数据类型，现在我们再来看一下变量在程序中的用法和变量在程序中扮演的角色。

当创建一个变量时，无论它是简单的整形变量还是复杂的派生类，用户必须考虑如何使用它们、哪个进程需要访问该变量和应该为这些变量提供什么样的保护级。

访问给定变量的能力取决于两个因素：创建变量时使用的修饰符和变量声明在类中的位置。

参见第 55 章的“字符—赋值”部分。

类的域与方法的变量的比较

在类中有两种类型的变量：直接属于类的变量和属于具体方法的变量。类中在任何方法之外声明的变量(一般位于类声明之后，方法声明之前)被称作类的域，域可以被类的所有方法访问。

另外也可以在方法中声明变量，这些变量是该方法的局部变量，局部变量只能在方法内访问。

因为方法的变量与方法的生命周期相同，所以它们不能够被其它的类访问，用户不能对方法变量使用任何访问修饰符。

虽然可以使每个域都能被每个类访问，但这不是精明的办法。首先，应该考虑使用类创造程序的目的。为什么要为类选择适当的类名而不使用 class 1、class2、class3 等来为类命名呢？因为用适当的名字为类命名可以使程序更加清晰易读，编码也容易，跟踪和调试也方便。由于相同的原因，所以要通过创建不同的保护级将代码封装在自足的逻辑块中。

另外，因为 OOP 严重地依赖对已编写过的代码的修改，访问限制可以防止用户以后做一些不该做的事情。(请记住：防止对域的访问并不是防止使用它)。例如，如果用户正在创建一个 Circle 类，可能要使用几个域来跟踪类的属性，如 radius(半径)、area(面积)、bordor _ color (边色)等等，这些域有许多是存在互相依赖关系的。虽然看起来将 radius 域设为 public(可被所有其它类访问)很合乎逻辑；但是如过上几周后再来编写如程序清单 11.9 所示的代码，看会发生什么事情：

```
import java.awt. * ;
class Circle {
  public int radius, area;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public Color border-color;
...
}
class GraphicalInterface {
Circle ball;
...
void animateBall() {
    for (int update-radius = 0; update-radius <= 10; update-radius++){
        ball.radius = update-radius;
        paintBall(ball.area, ball.border-color);
        ...
    }
}
void paintBall(int area,Color color){
    ...
}
}
```

这段代码将不产生用户期望的结果。虽然

ball.radius = update_radius ;

语句改变了半径,但它不会影响 area 域。由于这个原因,当使用 paintBall()方法时会得到不正确信息。但是,如果半径变量和面积变量是保护型的,并且在修改半径时强迫面积重新计算,那么上述问题就不会再出现了,如程序清单 11.10 所示。

```
class Circle {
    protected int radius, area;

        public void newRadius (int rad){
        radius = rad;
        area = rad * 2 * Math.PI;
    }

        public int radius(){
        return radius;
    }

    public int area (){
        return area;
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
class GraphicalInterface {
Circle ball;
...
void animateBall() {
  for (int update-radius = 0; update-radius < = 10; update-radius ++ ){
    ball.newRadius (update-radius);
      paintBall(ball.area(), ball.border-color);
          ...
          }
      }
  }
```

在下面几部分中,用户可以了解几种规范的访问方法并用它们解决现在这个问题。

虽然域的访问级很重要,其实域和方法变量在类中可见也是很重要的。变量可被访问的所有地方叫做范围。一般来说,每个变量只有在声明它的块内(限定在花括号{和}之内的代码)是可访问的。但是也有例外。请看程序清单 11.11 中的代码。

```
class  CashRegister {
    public int total;
    int sales-value[];
    Outputlog log;

    void printReceipt(int total-sale) {
        Tape.println("Total Sale = $ " + total-sale);
        Tape.println("Thank you for shopping with us. " );
    }

    void sellItem(int value) {
        log.sale(value);
        total + = value;
    }

    int totalSales() {
        int num-of-sales, total = 0;
        num-of-sales = log.countSales();

        for (int i = 1; i < = num-of-sales; i++)
            total + = sales-value[i];
        re
turn(total);
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

现在看一下例子中的某些变量和它们的范围。

变量名	声明名	范围
total	CashRegister 类的域	整个类
total	totalSales()方法的局部变量	在 totalSales()方法内
log	CashRegister 类的域	整个类
value	sellItem()的参数	在 sellItem()方法内
i	在循环中的属于 total Sales()方法的局部变量	在 for 循环中

表中有几件事情需要注意。从最简单的变量 log 开始。log 是 CashRegister 类的域，所以在整个类中都是可视的。类(以及同一个包中的其它类)中的方法都可以访问 log。类似地，value 虽然被声明为参数，然而它是 sellItem()局部变量。虽然 sellItem( )中的所有语句都可以访问 value，但是不能被其它的方法访问。稍难理解的是 i 变量，i 变量不是在方法的开始处声明的，而是在 for 语句中声明的。像 log 和 value 只用在定义它们的块中一样，i 只用在 for 循环中。下面的示例中，i 变量被重复地创建(在本例中创建了 10 次)。

```
for(int x = 0 x < 10 ; x ++ ){
    for (inti = 0 ; i< num _ of _ sales ; i ++ )
.......
}
```

为了更好地理解它，现在将这段代码翻译成 while 循环，这样看起来更加清楚些。

```
{
int x = 0 ;//声明 x 变量并为它赋初值
while (x < 10 ){
} // 开始另一层循环
int  i = 0 ; //声明 i 变量并为它赋初值
while (i < num _ of _ sales ){
.....// 可以在循环中做任何事情
i ++ ; // 执行 i 变量加 1
    }
}
x ++ ;// 变量 x 加 1
    }
}
```

正如用户看到的那样，即使是一个看上去很简单的 for 循环，如果把它隐含的括号都加上的话，就可以看到变量的范围实际上是比较复杂的。

最后我们看一下在重叠的范围里的两个 total 变量。既然 total 域可以被所有方法调用，那么在 totalSales()方法中就有问题了。在这时使用多定义标识符引用的是最里层的变量定

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

义。因此,虽然不影响类的其它部分,但在 totalSales()方法中标识符 total 引用的是局部变量 total,而不是全局变量 。这意味着在 totalSales()方法退出后,类变量的值不变。在这种情况下用户也可以通过使用 this 关键字访问类。如果要将类变量的值设置成局部变量的值,可以使用下述代码:

```
this.total = total ;
```

虽然使用相同的标识符作为域和方法名变量不会引起什么问题,是可以接受的做法,但是最好选用不同的(并且更具描述能力的)标识符,比如 total _ sales。另一种使代码易读的有效办法是为所有的类变量制定一个统一的命名方案。下面是两个普通的作法:

- 在所有类的域变量前加一个 m 作为变量的首字符
- 在所有域变量前加一个下划线(_)
- 从个人角度讲,作者更喜欢使用第二做法,因为它不会引起混乱。如果我们在前面的例子中使用这种命名方案,将会变成下面的样子:

```
class CashRegister {
    public int _total ;
    int _sales _value [ ]
    Outputlog _log ;
```

注意:虽然用户可以在方法中使用与域相同的标识符,这并不是适用于程序中所有代码块的。例如,如将 num _ of _ sales 声明为 for 循环的计数器,将会产生错误。

11.8.1 变量修饰符

像类修饰符和方法修饰符一样,变量的访问修饰符定义了某个变量可被其它的类访问的程度。对类的全局域使用访问修饰符很重要,而对方法里的局部变量使用访问修饰符几乎没有意义,因为方法的局部变量只生存在方法被执行的期间,当方法执行完毕后,这些局部变量被回收,并释放它们所占的内存,

---

为什么不把所有的变量都声明为域?

因为所有的类变量(域)可以被给定类的所有方法访问,那么为什么不把所有的变量声明为所有方法都可以访问的全局域呢?

第一个原因是这样做将浪费大量的内存。局部变量(在方法内部声明的变量)只存在于方法运行期间内,而域变量则存在于对象的生命期内。使用局部变量可以重复使用同一块内存。

第二个原因是把所有的变量都声明为全局变量会使程序难以阅读。如果仅需要在一个方法中使用计数器,为什么不把计算器声明在方法之内呢?而且,如果所有的变量都是全局变量,那么其它人在阅读程序时(或者用户本人在几个星期之后阅读该程序时),会很难找出这些变量是在什么地方赋值的,因为难以找出方法之间传递值的逻辑。

11.8.1.1 default

缺省情况下域可以被同一个包中的其它类访问,但不能被当前类的子类访问,也不能被其它包中的类访问。例如:

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

int　size

11.8.1.2　public

与方法的 public 访问修饰符一样，允许被修饰域在所有的类中可视，包括其它包中的类和所有的子类。另外用户应当努力限制使用 public。例如：

public int size ;

11.8.1.3　protected

protected 域可以被当前类的所有子类访问，但是在当前包以外的类中不可见。例如：

Protected int size ;

11.8.1.4　private

最高的保护级，private 域可以被当前类中的所有方法访问，但是不能被任何其它类访问，即使是当前类的子类也不可访问。例如：

private int size ;

11.8.1.5　static

和声明方法时一样，把修饰符放在域前即可以将域声明为静态域。static 域的值与该类所有实例中该域的值相同。也就是说当修改某个类中的某个 static 域时将影响该类的所有实例中的该域。static 域可以在 static 方法中修改，也可在 non _ static(非静态)方法中修改。

static int size;

参见第 8 章“方法”。

11.8.1.6　final

虽然 Java 没有 define 预处理语句，但是 Java 仍可以定义常量，即在程序运算期间不能修改的量。通过将修饰符 final 放在域前面可以告诉编译器该域的值在运行期间不能修改。因为不能再另外修改，所以用户必须在声明的同时为 final 域设置实际的值。例如：

final int SIZE = 5 ;

既然值不能修改，那么为什么不在程序中使用值本身呢？这个问题的答案是：

- 尽管不能在程序执行过程中修改常量的值，但是程序员在修改常量的值时可以不必修改多处代码。例如 SIZE 常量在代码中用过 10 次(10 个位置)，使用常量定义时；只要修改一个位置就可以把其它位置上的值全部修改。
- 通过使用常量，可以使代码更加清晰易读。例如，在 GameBoard 类中，如果使用 0 作为空位置检查，不如使用指定值为 0 的 final 域 EMPTY 那样更容易阅读和理解。

> **注意**：作为约定，常量的所有字都大写。为了节省内存，常量通常被声明为静态的。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:域还有两个附加的修饰符。在处理多线程时,如果多线程试图同时访问同一块数据,可能会发生问题。虽然这些问题的大部分可以通过方法同步来解决,但是在将来的Java版本中,用户可能要用到 threadsafe 域声明,Java 运算环境对这种域处理得格外小心。特别是每个易变的域在使用前后都会被检查。

另一个预言的关键字是 transient ,transient 是与创建持久 Java 小应用程序和持久 JavaBeans 的能力紧密相关的。在这种环境里,持久域不是持久对象的一部分。

11.8.2 使用方法来提供受保护的访问

虽然类中的某些域加以访问限制是一大优点;但是,为这些域提供某种方式的访问也是必要的。非常聪明和有效的作法是通过特定的公共方法(一般叫做 set _和 get _方法)来访问这些受限制的类,如程序清单 11.12 所示。

```
class Circle {
    private int radius, area;
    private Color border-color;

    public void setRadius(int update-radius) {
        radius = update-radius;
        area = Math.PI * radius * 2;
    }

    public Color getColor() {
        return(border-color);
    }
    public int getRadius() {
        return(radius);
    }
    public int getArea() {
        return(area);
    }
}
class GraphicalInterface {
    Circle ball;
    ...
    void animateBall() {
    for (int update-radius = 0; update-radius < = 10;
                        update-radius++) {
            ball.setRadius(update-radius);
            paintBall(ball.getArea(), ball.getColor());
        }
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
    ...
  }
}
```

通过限制 setRadius ( )方法对 radius 域的访问，可以保证 area 变量随着 radius 的修改而修改。因为已经将这两个域定义成了私有类型，所以用户必须自己通过各种 get_方法定义访问它们的手段。这些方法通常叫做访问器方法，因为它提供了对不可访问域的访问方法，这是在 Java API 库中极为流行的方法。

11.8.3　使用 finalize()方法

finalize()方法属于 Java.lang.Object 类，所以它可以被所有的类调用。缺省情况下这一方法是空的，Java 运行系统在无用单元回收过程中调用这个方法，用于在对象被撤销之前清除所有正在执行的进程。例如在处理套接字的类中，在撤销由该类定义的对象之前关闭套接字是一种好的做法，所以用户可以在 finalize()方法中放置关闭套接字的代码。当程序不再使用该类并需要撤销它时，这个方法就会被调用，用以按照要求关闭套接字。

> **注意：**Java 的 finalize()方法非常类似于C++中的~classname()方法。

接下来看一下程序清单 11.13 中的 finalize()方法。

```
import java.io.*;
Import java.net.*;
public class NetworkSender
{
    private Socket me;
    private OutputStream out;

    public NetworkSender(String host, int port) {
      try {
        me = new Socket(host,port);
        out = me.getOutputStream();
      }
      catch (Exception e) {
        System.out.println(e.getMessage();
      }
    }

    public void sendInfo(char signal) {
        try {
            out.write(signal);
            out.flush();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            }
            catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }

        public void disconnect() {
            System.out.println("Disconnecting...");
            try {
                me.close();
            }

            catch (Exception e)
            System.out.println("Error on Disconnect" + e.getMessage());

        System.out.println("done.");
    }

/* In this case finalize() is the identical to disconnect() /*
/* and only attempts to ensure closure of the socket in the /*
/* case that disconnect() is not called. */

    protected void finalize() {
        System.out.println("Disconnecting...");
        try {
            me.close();
        }
        catch (Exception e)
            System.out.println("Error on Disconnect" + e.getMessage());

        System.out.println("done.");
    }
}
```

注意:finalize()在 java.lang.Object 中声明的是 protected 型的,因此只能使用 protected 或低于 protected 的修饰符来声明。

警告:尽管 finalize()方法是一个合法的工具,但是不能过分依赖这个工具,因为无用单元回收不是一个完全可以确定的过程。无用单元回收作为低优先级的线程在后台运行,并且往往是在没有内存可用时才调用它。

因此最好单独编写用于清理内存的代码,而将 finalize()作为最后的备用手段。这样即使是自己编写的代码不能运行也不会引起严重的问题。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

11.9　嵌入类

在 Javal.1 编译器中，Sun 公司为语言增加了几个新特性。其中之一是嵌入类。嵌入类只能被 Javal.1 和 Javal.2 编译器编译。

11.9.1　什么是嵌入类

嵌入类实际上是包含在另一个类的体中的类。事实上，用户甚至可以在方法的体中包含类。嵌入类对于程序员是非常有用的，因为它们可以使用户以更易管理的方式来构造代码。另外，在某种情况下，它们可以增强代码的可读性。

用户可能对使用嵌入类感到奇怪。因为使用嵌入类不需要用户开发任何东西。使用嵌入类可以为用户提供组织更易理解的代码的能力，并且有时也可以为编译器提供进一步优化最终代码的手段。使用嵌入类可以取得与普通类一致的结果。

从这一点来说，如果用户是一位经历了C++发展进化的程序员，用户可能会认为嵌入类对设计者来说是一个好主意。但最后却发现它是一个容易引起混淆的概念。Java 是不是要避免这种混淆呢？避免运算符覆盖，多继承和其它像在C++这样的语言中被认为是好用但却容易混淆的不合理做法吗？然而不幸的是：答案是肯定的。时间可以告诉所有的开发群体，完整地接受嵌入类是很有好处的。不管用户本人的观点如何，理解嵌入类是很重要的，以防万一正在编辑的代码是别人利用嵌入类功能实现的。带着这种思想继续向下阅读，看看嵌入类究竟是怎样工作的。

11.9.2　使用嵌入类创建程序

嵌入类的主要优点是能够创建适配器类。适配器类是用于实现接口的类。通过分离，将独立的适配器纳入到嵌入类中，用它基本上可以在一个顶级类中建立一个类似于包的结构。

看一下程序清单 11.14 中使用适配器类的例子。这个例子展示了如何用同一个类创建两个独立的 runnable 接口。这两个接口都需要访问顶级类的 currentCount 变量。

```
/*
 *
 * BigBlue
 *
 */
public class BigBlue implements Runnable{
    int currentCount;

    class Apple implements Runnable {
        public void run(){
            while(true){
                System.out.println("count = " + currentCount);
                try{
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                Thread.sleep(100);
            }catch (Exception e){}
        }
    }

    public Runnable getApple(){
        return new Apple();
    }

    public void run(){
        while(true){
            currentCount += 5;
            try{
                Thread.sleep(75);
            }catch (Exception e){}
        }
    }

    public static void main(String argv[]){
        BigBlue b = new BigBlue();

        Thread appleThread = new Thread (b.get Apple());

        appleThread.start();
        Thread thisThread = new Thread (b);
        thisThread.start();
    }
}
```

当用户阅读前面的例子时，注意 BigBlue 类的 run()方法直接访问了 currentCount 变量，currentCount 是 BigBlue 类的域。它的作用就像其它任何方法的作用一样。现在看一下 Apple 类。这个类也访问了 currentCount 变量，并且就像访问自己的域一样，只有一点不同，那就是它是从顶级类 BigBlue 接收的。

为了编译这个程序不必将 Apple 和 BigBlue 两个类全都编译，只要编译 BigBlue 类就可以：

javac BigBlue.java

为了运行这个程序，可以输入下列命令：

java BigBlue

用户最后看到的是一个数字序列。注意：因为 BigBlue 线程的休眠时间比 Apple 略短，所以前者的数字增长比后者快。这个程序实际上有两个线程，并且运行了两个完全不同的循环。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

警告:如果在编译含有嵌入类的类时发现类似下面的错误:

bigBlue.java :30 :

no enclosing instance of class bigBlue is in scope ; an explicit one must be provided when creating class bigBlue.apple,as in outer. new inner( ) or outer.super ( )

Thread appleThread =new Thread (new apple( ))

用户可能非常困惑。为了解释这个错误,先看一下 main 方法是如何产生这个错误的:

```
public static void main (String argv[ ])
bigBlue   b = new bigBlue ( ) ;
Thread appleThread =new Thread (new apple ( ) );
appleThread.start ( ) ;
Thread thisThread =new Thread (b) ;
thisThread .start ( );
}
```

导致这一错误的原因是试图在静态的 main 方法中创建一个新的 apple ()实例。为了能够访问 apple 类,必须使用非静态的 BigBlue 实例。

11.9.3　同步嵌入类

在任何时候同步嵌入类的父类的方法都是很重要的。一般情况下,可以声明某个方法是 synchronized 的,或者创建一个 synchronized(this)块。因为 this 是新类,那么如何为父类指定同步呢? 考察一下程序清单 11.15 中创建枚举数的情况。

```
public class FixedStack {
int array[ ] = new int[10];
      int top=0;

synchronized public void push(int item) {
array[ top++ ] = item;
            }

            class Enumerator implements java.util.Enumeration {
public int nextElement( ) {
                        synchronized (FixedStack.this) {
                              if (count > top)
                              count = top;
                              if (count = = 0)
                              throw new NoSuchElementException("FixedStack");
                              return array[ - - count];
                  }
                  }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:在程序清单 11.15 中,不要试图在使用 push( )方法向数组添加元素的同时使用 Enumeration 类的 nextElement ( )方法访问数组中的元素。为了访问数组中的元素,该例使用了有资格的 FixedStack.this 变量。有资格类指的是 this 的超类。嵌入类隐含地知道有资格的 this 指的是嵌入类的实例。

11.9.4 嵌入类是如何工作的

现在,用户可能还不知道嵌入类是如何工作的。在 Java1.0 版本中没有提供嵌入类。那么如何能让使用嵌入类编写的程序在 Java1.0 版本中的虚拟机上运行呢? 答案就在于嵌入类实际上不是什么新东西。当使用嵌入类编写类时,编译器将嵌入类从主类中分离出来并调整编译结果。

如果用户是一位经历了早期C++修改发展的程序员,应该知道当时的嵌套类产生的是一个提示。原因是刚刚问世的C++是实际是穿着面向对象外衣的 C 语言。当用户编写C++程序时,C++编译器只是将C++代码转化为 C 代码,然后由 C 编译器执行实际的编译。而 Java 不需要两个编译器,但仍然可以取得极为相似的结果。

11.9.5 为什么使用嵌入类

用户可能对自己说:"为什么我要使用嵌入类呢?"。答案是为了使用更合适的方式组织代码。Sun 公司的文档称这些嵌入类的为适配器(Adapter)类。为了理解这个名词先看一下它的用途。

嵌入类可以扩展成或应用为任何用户想要的接口(普通的类也能实现)。唯一的问题是当使用标准类实现接口时,很难定位与接口关联的方法在代码中的位置。因为嵌入类的声明和代码是放在一起的,所以用户看起来非常清楚。

11.10 包

当用户开始为程序创建大量的类时,把它们放在一起很重要。一堆各式各样的类文件就像没有文件夹或子目录的硬盘一样是杂乱无章的。想象一下如果把硬盘里的所有文件都放在一个文件夹下将是怎样的情形。用户可能有几千个文件,那么这几千个文件就不能有重名的。

类文件也是按照相同的方式存放的。同文件夹下的文件不能重名,这是硬性要求。为了克服这一点。Java 提供了一个名为包的系统。用户可以认为每个包都是一个子目录或文件夹。Java 用户已经看到 Java API 中使用了大量的包,java.awt 是一个包,java.lang 是另一个包,等等。

Java 中的包是一组类,这些包与许多计算机语言的库很类似,典型的 Java 包一般都包含着相关的类。用户可以想象一个名叫 Transportation(运输工具)的包,这个包可能定义大量的诸如 Car(汽车)、Boat (轮船)、Airplane(飞机)、Train(火车)、Rocket (火箭)、Amphibiuscar(水

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

陆两栖战车),SeaPlane(水上飞机)等这样的类,当应用程序在处理这类项目时,可以通过从 Transportation 包中引入类来提高开发效率。

要想使某个类成为包的成员,必须使用 package 语句声明它,例如:

```
package Transportation;
```

package 语句有几个特殊的要求:

- 对于将要包含到包中的类,要求它的代码必须和包中其它文件在同一个目录下。用户可以规避这个要求,但这不是好主意。
- package 语句必须是文件中的第一个语句。换句话说,在 package 语句之前除了空白和注释以外不能有任何东西。下面的表显示了一个合法的 package 语句和一个不合法的 package 语句:

合法的	不合法的
package Transportation ;	import java.applet.Applet
import java.applet.Applet	package Transportation ;

11.11　引入包中的类

将文件声明为包的一部分之后,实际类的名字应该是包名加点(.)加类名。换句话说,在 Transportation 例子中的 Car 类应为 Transportation .Car 类,在此之前它还是简单的 Car 类。

这可能会引发一个容易解决的小问题。如果用户写了一个程序,后来决定将它定义成某个包的成员,那么编译器将如何寻找其它文件呢? 在此之前它们叫做 Car 和 Van,而现在必须使用 Transportation.Car 引用它们才能够使用它们。换句话说,以前引入 Car,而现在必须引入 Transportation .Car; 如下所示:

老代码	**新代码**
import Car	import Transportation.Car

11.12　引入整个包

用户可以引入包的全部内容或包中的所有类。用户可能已经看到了使用 JDK 类时曾这样做过,比如 java.awt 。为了引入所有的类,可以使用通配符(*)代替各个类名:

```
import java.awt.* ;
```

通过引入整个包,用户可以使用包中的每一个类。这可能是很方便的,因为用户可以减少许多输入工作。例如:

```
import java.awt.Graphics;
import java.awt.Image ;
import java.awt.Button ;
import java.awt.Canvas ;
...
```

现在,是否用户正在想,"既然简单,为什么不每次都引入整个包呢?",答案是否定的,因为引入整个包有几个弊端:

- 当用户引入整个包时,虚拟机必须跟踪包中所有元素的名字。使用额外的内存存储类

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

和方法名不再是特别重要的问题了，因为现在用户使用的计算机可能有 16M 或 16M 以上的内存了。总之随着越来越小的基于 Java 的计算机出现，这可能成为一个问题。另外这将会稍稍降低一点计算机的效率。

- 如果用户引用了几个包，并且它们有共享的文件名，系统就要崩溃了。究竟想要哪个类？例如，如果用户引入了 YourCorp.*，该包中有一个 Button 类，又引入了 java.awt.*，该包中也包含一个 Button 类，这两个 Button 类将引起冲突。
- 最重要的弊端涉及到了国际互联网的带宽问题。当引入不在本地机器上的整个包时，(没有包括在 java.* 包内)，小应用程序浏览器或其它的浏览器必须在继续执行它之前通过网络将该包的所有文件都拖过来。如果包中有 30 个类，但是只使用两个，小应用程序就不能尽快地加载，所以用户将浪费许多资源。

11.13 不通过引入使用类

在此之前用户可能还没有意识到，在使用之前引入类实际不是必要的。一般情况下，null 包中的类和存放在同一个物理目录中的其它的类可以不经任何操作即可以互相引用。例如，如果有两个类 Car 类和 Van 类在同一个目录中，用户可以在 Van 类中直接创建 Car 类的实例，而不必引用 Car 类。程序清单 11.16 和程序清单 11.17 展示的就是这样两个示例：

```
//Car is just a generic class with a few variables
public class Car {
  int wheels;
  int tires;
  int speed;
  //simple constructor
  public Car (int inWheels, int inTires, int inSpeed){
    wheels = inWheels;
    tires = inTires;
    speed = inSpeed;
  }
}
```

```
//The Van class is another simple class, but uses the Car class
public class Van {
  //The Car class is used here without being imported
  Car theCar;
  int doors;
  //simple constructor
  public Van (Car inCar, int inDoor){
    theCar = inCar;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        doors = inDoor;
    }
}
```

当用户将某个类放进某个包时，可以继续不经引入而使用它。唯一的不同是必须使用完整的类名来声明实例。程序清单 11.8 和 11.9 除了 Car 类是 Transportation 包的成员这一点不同外与程序清单 11.5 和 11.6 是一致的。

程序清单 11.18　包中的 Car 类

```
package Transportation;
//Car is just a generic class with a few variables
public class Car {
    int wheels;
    int tires;
    int speed;
    //simple constructor
    public Car (int inWheels, int inTires, int inSpeed){
        wheels = inWheels;
        tires = inTires;
        speed = inSpeed;
    }
}
```

程序清单 11.19　使用包中的 Car 类

```
//The Van class is another simple class, but uses the Car class
public class Van {
    //The Car class is used here without being imported
    Transportation.Car theCar;
    int doors;
    //simple constructor
    public Van (Car inCar, int inDoor){
        theCar = inCar;
        doors = inDoor;
    }
}
```

注意：虽然不经引入包也可以使用包中的类，但必须指明类是定义在该包中。在前面的例子中，如果引入了包：

```
import Transportation.Car ;
```

那么在建立 Car 类的对象时就不必在每次引用 Car 时在 Car 前面加 Transportation 了，那么代码就与程序清单 11.18 中的代码一致了。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

11.14 使用包来组织代码

包不仅仅是捷径,它们还能使程序井井有条。Java 本身就是一个内置的包集。

表 11.1 标准的 Java 包

包	描 述
java.applet	包含一些用于创建 Java 小应用程序的类,这些小应用程序可以在像 Netscape 2 .0(或更高版本)、HotJava 和其它与 Java 兼容的浏览器上运行
java.awt	包含一些用于编写平台无关的图形用户界面(GUI)应用程序的类。它包含几个子包,包括 java.awt.peer 和 java.awt.image
java.io	包含一些用作输入输出(I/O)处理的类。数据流就包含在这里面
java.lang	包含一些基本 Java 类。java.lang 是被隐式地引入的,所以用户不必引入它的类
java.net	包含用于建立网络连接的类。与 java.io 同时使用以完成与网络有关的读和写
java.util	包含一些其它的工具和数据结构,如编码、解码、向量和堆栈等

另外还有一些包是针对商业用途的。

从包的特点可以看出 Sun MicroSystem 公司是如何使用包来将相似的类组织在一起的。当用户开始构造一个程序时,可能会设想把整个程序放在一个包中。比如说用户要编写一个超人游戏。用户可以把所有的类都放在名为 Pac 的包中。这是不是好主意? 可能不是,这完全取决于实现方式。

可以肯定地说在超人游戏中将包含许多用户以前写的打斗类游戏中使用的代码。例如,建一个精灵产生器。把所有用于实现精灵的元素放在一个包中,而把超人游戏中特有的东西放在 Pac 包中是一种有远见的作法。以后用户可以为精灵包增加内容,这样做不会影响超人游戏的可读性。

11.15 所有 java.lang 类的隐式引入

通过阅读本书的代码,用户不难发现,Java 有一套不需引入就可以直接使用的类。java.lang 包中的类总是已经引入了的。这就意味着用户可以将 java.lang.Sytem 类作为系统的一部分来使用,不必在语句中引入,例如:

```
System.out.println("Hi there,System hasn't been imported ")
```

当使用 System 时,这是非常方便的,但这也是引起混乱的祸根。例如,在程序清单 11.10 中;Math 类也是在未被实例化的情况下访问了 PI 的值的。如果不熟悉 Math 是如何被(自动地)引入的,就可能引起疑惑和不解。克服这个理解困难的办法就是熟悉 java.lang 包并掌握其中的类的用法。

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

第 12 章　接　　口

本章内容：

12.1　什么是接口
12.2　创建接口
12.3　使用接口
12.4　从其它类中使用接口
12.5　异常

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

12.1 什么是接口

接口是对C++的多继承性的替代，通过接口某个类可以拥有几个超类。虽然希望一个类能够继承多个类的特性，但是在考虑到几方面的原因后，Java的缔造者们还是决定禁止多继承性。Java类可以使用多个接口，这样就可以在避开多继承性问题的前提下基于其它类来建立类。

在语法上接口与类相似，接口用于定义几个类具有的但又不是在这些类中定义的功能。通过在接口中设置这种方法，用户可以概括出共同的行为，并留给类本身去具体实现。在高级的数据处理中使用接口是比使用类更好的选择。

接口是类的低层次的亲戚。事实上，它特别像纯粹的抽象类。虽然类具有定义对象的能力，而接口定义的是供别的类使用的方法和常量。从实际的观点看，接口通过为对象声明一些特性来帮助对象定义行为。例如，知道某个人是运动员并不能定义她的全部个人的特性，但是可以肯定她具有一定的品质和能力。

作为例子，假设运动员能够在跑100米的时间里跑完一英里，并且还能够举起重物。通过使用运动员接口后，用户可以保证某个人拥有这些能力。

以另一种方式来考虑一下接口，想一想收音机、电视机和计算机喇叭。他们都有共同的控制音量的特性，那么我们就可以让这些设备使用一个名为VolumeControl的接口。

接口有一个主要限制：它们可以定义抽象的方法和最终的域，但是不能具体实现这些方法。对于方法，这意味着体是空的。使用接口的类负责具体实现这些方法。这意味着与扩展类不同，在使用界面时用户必须覆盖接口中的每个方法。从一般意义上讲，接口允许用户为程序员设计一些功能，但不必关心这些功能以后如何具体地定义。例如，如果某个类使用了java.lang.Runnable接口(有一个run( )方法的接口)，那么这个类就知道该接口有run( )方法。因为虚拟机能够保证Runnable类有run( )方法，所以虚拟机可以盲目地调用run( )方法。另外当语言设计者们在编写虚拟机程序时，他们并不知道在run( )方法内将发生什么。所以用户即可以做动画，也可以计算前1000个质数，这都没有关系。有关系的是用户必须建立和运行Runnable接口。

另一个优秀的示例是java.applet.AppletContext接口。这个接口定义了一套方法用于返回与小应用程序运行的环境有关的信息。例如，AppletContext定义了一个名为getImage的方法。通过这个方法浏览器可以使用小应用程序加载图像。

问题是不同的浏览器如AppletViewer或Netscape Navigator加载图的方式不同。更糟糕的是，即使是同一个浏览器在不同的平台上工作方式也有差异。幸运的是每一个支持Java的浏览器有AppletContext接口，这样一来，虽然java.applet.Applet类依赖于在AppletContext接口中声明的方法，但是用户不必关心这些方法是如何工作的。这也就是说用户可以在不同的环境和浏览器里使用相同的小应用程序和相同的方法(比如java.applet.Applet.getImage())，但用不着担心哪里有getImage( )方法。

12.2 创建接口

创建接口的语法与创建类的语法相似。但也有些不同。最大的差别是接口中没有一个方

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

法拥有方法体，并且也没有除常量外的任何变量。

程序清单 12.1 是一个接口的示例。它展示了三项内容：接口、使用接口的类和使用派生类的类。通过这个例子了解一下如何使用接口，就能猜测出本章要讲的内容了。

```
public interface Product {
    static final String MAKER = "My Corp";
    static final String PHONE = "555 - 123 - 4567";

    public int getPrice(int id);

}

* * * begin Listing 12.1a: Shoe.java: Class Shoe which implements the Product
Interface
public class Shoe implements Product {
    public int getPrice(int id) {
    if (id = = 1)
      return(5);
    else
      return(10);
    }
    public String getMaker() {
      return(MAKER);
    }
}
* * * Begin Listing 12.1b: Store.java: Class Store extends Shoe(which implements
Product)
public class Store {
    static Shoe hightop;

    public static void init() {
      hightop = new Shoe();
    }

    public static void main(String argv[]) {
      init();
      getInfo(hightop);
      orderInfo(hightop);
    }

    public static void getInfo(Shoe item) {
      System.out.println("This Product is made by" + item.MAKER);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        System.out.println("It costs $" + item.getPrice(1) + '\n');
    }

    public static void orderInfo(Product item) {
        System.out.println("To order from" + item.MAKER + "call" + item.PHONE + ".");
System.out.println("Each item costs $" + item.getPrice(1));
    }
}
```

12.2.1 声明

声明接口的语法是：

public interface 接口名 *extends 接口列表*

在这里，斜体部分是可选的。

12.2.1.1 public 接口

在缺省情况下，接口可以被同一个包中的所有类访问。但是如果使用 public 声明了接口，那么其它的类和对象也可以访问该接口。

提示：就和公有类一样，公有接口必须定义在与接口同名的文件中，即“接口名.java”文件中。

12.2.1.2 接口名

接口的命名规则与类的命名规则是一样的。唯一的要求是必须使用字母，下划线和货币符号开头，名字中只能包含统一码字符（基本字符和数字字符及某些特殊字符）；另外不许与 Java 关键字同名（如 extends 或 int）。将接口名首字符大写是一种习惯做法，不是语法要求。

参见第 6 章的“关键字”部分。

提示：虽然只是针对公有接口要求的，实际上别的接口也可以为其它所有接口按照“接口名.java”方式命名，这样做可以使用户自己和编译器很方便地查找类的代码。

也就是说，即使 Product 接口不是公有的，用户也应该把它声明在 Product .java 文件中。

12.2.1.3 扩展其它接口

把 OOP 的继承性作为一种手段，可以把已经开发好的接口的代码扩展成更大的子接口。新的子接口继承了超接口的全部方法和全部静态变量，就像子类继承超类的属性一样。

参见第 5 章的“面向对象编程：新的思维方式”部分。

在扩展其它接口的过程中需要掌握的原则是不能定义父方法的体，也不能定义自己的方法体。任何使用新接口的类必须定义父接口和子接口的所有方法体。

在下面的示例中，新接口扩展了前面定义的接口（Runnable）：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
interface MonitoredRunnable extends java.lang.Runnable {
  boolean isRunning ( );
  }
}
```

这个声明显示了一个更详细的接口，包括了一些 java.lang.Thread 的属性。

> **注意**：接口不能扩展类。不能这样做有许多原因。最容易理解的是要被接口扩展的类应该已经拥有自己定义好了的方法体。这恰恰是与接口定义相违背的。

记住，如果使用了扩展的接口，必须将老接口的方法和新接口的方法统统地覆盖掉。如程序清单 12.2 所示。

```
class Fireworks implements MonitoredRunnable {
    private boolean running;        // Keeps track of state.

    void run() {
      shootFireWorks();
    }
    boolean isRunning() {        // Provides access to other objects without
      return(running);      //allowing them to change the value of running.
    }
}
```

因为 Fireworks 使用了 MonitoredRunnable，所以必须覆盖 isRunning ( )方法。IsRunning ( )方法是在 MonitoredRunnable 中声明的。因为 MonitoredRunnable 扩展了 Runnable，所以也必须覆盖 run()方法，run()方法是在 Runnable 中声明的。

> **注意**：类是利用接口来继承几个类的属性的，即接口扩展了接口。当需要扩展多个接口时，可以使用逗号将它们分开。这意味着，虽然类不能扩展多个类，但是接口可以扩展多个接口。

```
interface MonitoredRunnable extends java.lang.Runnable ,java.lang.cloneable {
  boolean isRunning ( );
  }
}
```

12.2.2　接口体

接口体不能指定任何方法的具体实现，但是可以指定它们的属性，另外接口中可以包含 final 变量。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

例如,在 Product 接口中声明的 MAKER 变量名允许用户声明一个可供所有使用 Product 接口的类使用的常量。

另一个比较好的例子是 java.awt.image.ImageConsumer 接口中的 final 域。该接口定义了一套 final 整数。这些整数用作解释信息的标准。因为 RANDOMPIXELORDER 变量等于 1,使用 ImageConsumer 接口的类可以引用该变量,并知道 1 的含义是按照随机的顺序发送像素。程序清单 12.3 展示了 setHints 方法 。

```
public class MagnaImage implements ImageConsumer{
    imageComplete(int status) {
      ...
    }

    setColorModel(ColorModel cm) {
      ...
    }

    setDimensions(int x, int y) {
      ...
    }
    setHints(int hints) {
      if ((hints & RANDOMPIXELORDER)! =0){
        ...
      }
    }

setPixels(int x, int y, int w , int h, ColorModel cm , byte pixels[],int off, int scansize) {
      ...
    }

setPixels(int x, int y, int w, int h, ColorModel cm, int pixels[], int off,int scansize) {
      ...
    }

    setProperties(Hashtable props) {
      ...
    }
}
```

12.2.2.1 方法

接口的主要用途是声明抽象的方法;这些方法将在引用该接口的类中定义。如果用户正在处理一个使用接口的类,那么这些方法就应该在该类中定义出来。虽然这个过程并不复杂,

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

但是有一个重要的差别值得注意。

在接口中声明方法的语法与在类中声明方法非常类似，但是与在类中声明方法相对比，接口中的方法不能拥有体。接口的方法的唯一组成部分是声明。例如，像下面的两个方法，如果它们是在接口中定义的，那么它们就是完整的：

```
Public int getPrice (int id ) ;
Public void showState ( )
```

但是如果在类中定义方法，那么就一定要求定义方法的体：

```
Public int getPrice (int id ){
if (id = = 1)
  return(5) ;
else
  return(10);
}
Public void ShowState ( ){
  System.out.println ("Massachusetts");
}
```

接口的方法声明并不决定方法的行为，只定义使用方法时需要的信息和方法将要返回的信息。接口的方法实际是在某个具有该接口的属性的类中定义的。所以在接口中定义方法主要是确定返回类型和参数表。

接口中的方法声明具有如下所示的语法：

public　返回值　方法名(*参数*)　*throws*　*异常列表*；

这里的斜体部分是可选项，只有 public，返回值和方法名是必选项。与类中的方法不同，接口的方法在声明之后就加上语句结束符分号。

> **注意**：界面中的方法在缺省情况下都是公有的，不管使用不使用 Public。
>
> 修饰符结果都是一样的。而类的缺省保护级是 friendly。另外当在接口中声明方法时使用其它的方法修饰符(包括 native，static final，Synchronized ，Private ，Protected 或 Private Protected)都是不合法的。

12.2.2.2　接口中的变量

虽然接口一般用于提供方法的抽象实现，但是用户可以在接口中定义实实在在的变量。因为不能在方法体中放置任何代码，所以全部声明在接口中的变量都是全局变量。换句话说，在声明域时，不管使用什么样的修饰符，这些在接口中声明的域都是公有的，最终的和静态的。

> **提示**：虽然所有的域都是以 public、final 和 static 形式创建的，但不必在声明中显式地使用这些修饰符。不管声明中出现不出现这些修饰符，域缺省地就拥有这些特性。所有在接口中声明的域都是公有的，最终的和静态的，请用户时刻牢记这一点。

正如在 Product 接口中看到的那样，接口的域就像类中的最终静态域一样，用于定义可被

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

所有使用该接口的类访问的常量：

```
public interface Product { //这个变量是静态的和最终的。
ptatic final String MAKER = My Corp ;
//虽然这个变量没有显式地说明,但是这个变量仍然缺省地是静态的和最终的。
String PHONE ="555 - 123 - 4567" ;
public int getPrice (int id ) ;
}
```

12.3 使用接口

既然用户知道了如何创建接口，现在让我们考察一下在开发类时如何使用它们。程序清单 12.4 展示了一个使用 Product 接口的类。

```
class Shoe implements Product {
    public int getPrice(int id) {
        if (id = = 1)
            return(5);
        else
            return(10);
    }
    public String getMaker() {
        return(MAKER);
    }
}
```

当然，类中的代码可以处理与接口无关的功能(如 getMaker( )方法)。但是为了满足使用 Product 接口的要求，该类必须覆盖 getPrice (int)方法。

12.3.1 覆盖方法

在接口中声明方法是个好做法。但是在使用接口和覆盖方法之前不能使用该方法。

提示：请记住，如果使用了接口，必须把声明在接口中定义的所有方法覆盖掉，否则该类就成了抽象类了。

12.3.2 修饰符

正如在前面讨论过的那样，声明在接口中的方法将被缺省地赋予公有访问级。因为不能为已有的方法通过覆盖来提高它的访问级，所以在接口中声明并在类中覆盖的方法必须被赋予 public 访问修饰符，除非它们已经在接口中被显式地声明为低于 public 的访问级。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

其它可用于方法的修饰符只有 native 和 abstract，它们可以用在接口中声明的方法中。

12.3.3　参数列表

接口方法中定义了一套参数，这些参数必须传递给方法。具有相同名字和不同参数的方法将重载该方法，但不会覆盖该方法。

虽然重载接口中的方法不是什么错误，但是使用在接口中声明的方法也是很重要的。所以，除非将类声明为抽象的，否则必须覆盖每个方法，并且要使用与接口相同的的参数签名（见程序清单 12.5）。顺便说一下，只有一个方法满足 run 方法对 Runnable 接口的要求。

```
public void Runner implements Runnable {

//This method overloads the run() method; it does not
//fulfill the requirements for Runnable.
public void run(int max){
    int count = 0;
    while (count++ < max){
        try{
            Thread.sleep(500);
        } catch (Exception e){}
    }
}

//This method fulfills the requirement for Runnable.
//You must have this method.
public void run(){
    while (true){
        try{
            Thread.sleep(500);
        } catch (Exception e){}
    }
}

}
```

如果方法 String CreateName (int length ,boolean capitalized)在接口中声明了，就有覆盖正确与否的问题，这两个方法中有一个是有效的，而另一个则是无效的，但是与接口无关：

有效的

String CreateName (int a ,boolean b)

String CreateName (int width ,boolean formatted)

无效的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
String CreateName (boolean capitalized ,int length )
String CreateName (int length )
```

12.3.4 体

当创建一个使用接口的类时,用户主要关心的是创建在接口中声明的体,除非用户决定将方法设成本地的。如果不想让新的方法成为抽象的,那么就应该为在接口中声明的方法创建体。虽然接口保证了在非抽象的类中所有的方法都有实在的定义并且都返回适当的数据类型,除此之外接口对方法的体没有什么进一步的限制。

12.4 在其它类中使用接口

到目前为止我们已经介绍了如何创建接口和基于接口的类。但是接口除了用于开发使用派生类和接口本身的类以外再没什么用处了。

12.4.1 使用接口的域

虽然接口的域必须是静态的和最终的,但它们在代码中是非常有用的。

下面的例子演示了接口中的任何变量可以通过在类中使用圆点来引用。这意味着用户可以使用 java.awt.image.ImageConsumer.COMPLETSCANLINES,就像使用 java.awt.Event 类中的 java.awt.Event.MOUSE_DOWN 常量一样,这样就提供了对接口中常量的访问。程序清单 12.6 演示的是使用 ImageConsumer 接口的另一个常量的例子。

```
import java.awt.image.*;
class MyImageHandler {
/* The java.awt.image.ImageConsumer interface defines certain constants to serve as indicators.
STATICIMAGEDONE, which is set to equal 3, informs the consumer that the image is complete.*/
    ImageConsumer picture;

    void checkStatus(boolean done) {
        if (done)
            picture.imageComplete(ImageConsumer.STATICIMAGEDONE);
        }
}
```

12.4.2 把接口当作类型使用

接口最重要的特性是它可以用作一种数据类型。接口变量可以被任何类使用。

12.4.2.1 用作参数类型

在程序清单 12.7 中,用户可以创建一个简单的使用前面开发的 Shoe 类的应用程序,因为 Shoe 类使用了 Product 接口,所以既可以将 Shoe 类的实例当作标准的 Shoe 对象来处理,也可

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

以将 Shoe 类的实例当作基于 Product 接口的对象来处理。虽然两种处理办法都可以得到相同的结果，但是将实例当作基于 Product 接口的对象处理可以为用户提供更多的灵活性。

```
class Store {
    static Shoe hightop;
    public static void init() {
        hightop = new Shoe();
    }

    public static void main(String argv[]) {
        init();
        getInfo(hightop);
        orderInfo(hightop);
        }

    public static void getInfo(Shoe item) {
System.out.println("This Product is made by" + item.MAKER);
        System.out.println("It costs $" + item.getPrice(1) + '\n');
}

    public static void orderInfo(Product item) {
        System.out.println("To order from" + item.MAKER + "call" + item.PHONE + ".");
        System.out.println("Each item costs $" + item.getPrice(1));
    }
}
```

12.4.2.2　输出

在下面的例子中 getInfo()方法将 hightop 作为带有一定方法和域的简单类来处理。有趣的示例是 orderInfo( )方法，它在不知道 Shoe 任何情况的前提下提取了几乎与 Shoe 完全相同的信息。因为 Shoe 满足了 Product 的要求，所以用户能够隐式地强制 Shoe 成为 Product。其结果是，因为用户知道 Product 接口声明了一定的特性，所以用户可以保证这些特性，比如 getPrice()方法，出现在参数项中：

```
c: \ dev  \ jdk \ java \ bin \ java Store
this Product is made by my Corp
It costs $ 5
To order from My corp call 555 - 123 - 4567.
Each item costs $ 5
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:注意当将 hightop 作为 Product 来处理时,用户可以隐式地将它强制为一种新数据类型,不必在代码中特别陈述。虽然编译器没有出错,用户还是有必要用下面的语句代替 Store 类中的 orderInfo(hightop)语句:

orderInfo ((Product) hightop);

这个语句完成的任务与原来语句完成的是同样的任务,但是它更容易被其它程序员读懂,因为它显示了 OrderInfo()接受的是 Product 作为参数,而不是 Shoe。

虽然在这样简单的例子中使用 Product 作为参数类型不是必须的,但是当有多个类都使用相同的接口时,这样做就显得格外重要了。如程序清单 12.8 中的例子,这个精心制作的 Store 拥有 7 个项目,这 7 个项目,都使用 Product 接口,如程序清单 12.8 所示。

```
interface Product {
    static final String MAKER = "My Corp";
    static final String PHONE = "555 - 123 - 4567";
    public int getPrice(int id);
    public void showName();
}
class Book implements Product {
    public int getPrice(int id) {
        if (id = = 1)
            return(20);
        else
            return(30);
    }
    public void showName() {
        System.out.println("I'm a book!");
    }
}
class Shoe implements Product {
    public int getPrice(int id) {
        if (id = = 1)
            return(5);
        else
            return(10);
    }
    public void showName() {
        System.out.println("I'm a shoe!");
    }
}
class store {
    static Shoe hightop;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    static Book using_java;

    public static void init() {
        hightop = new Shoe();
        using_java = new Book();
    }

    public static void main(String argv[]) {
        init();
        orderInfo(hightop);
        orderInfo(using_java);
    }

    public static void orderInfo(Product item) {
        item.showName();
        System.out.println("To order from" + item.MAKER + "call" + item.PHONE + ".");
        System.out.println("Each item costs $" + item.getPrice(1));
    }
}
Output:
C:\dev>\JDK1.2\java\bin\java Store
I'm a shoe!
To order from My Corp call 555-123-4567.
Each item costs $ 5
I'm a book!
To order from My Corp call 555-123-4567.
Each item costs $ 20
```

12.5　异常

为了使接口方法能够放弃异常，异常类型(或它的一个超类)必须出现在方法定义的异常列表中，就像定义在接口中一样。下面是覆盖放弃异常的规则：

- 新的异常列表只能包含原来异常列表中列出的异常或原来列出的异常的子集。
- 新的异常列表不需要包含任何异常，不管原来异常列表中列出异常的数量多少(这是因为原来列表被继承地赋给了新方法)。
- 新方法可以放弃在原来异常列表中列出的或在从原来列表中派生出的列表中列出的任何异常，与自己的异常列表无关。

通常，在接口中声明的方法的异常列表，在方法中就不再重复声明了，这个列表决定了什么异常可以放弃，什么异常不能放弃。换句话说，重复声明的方法改变了异常列表，它不能在原来接口声明中指定的异常列表里增加任何新异常。

最后看一下程序清单 12.9 替换异常列表。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
interface Example {
    public int getPrice(int id) throws java.lang.RuntimeException;
}

class User implements Example {
    public int getPrice(int id) throws java.awt.AWTException {
    // Illegal - Reason 1
    // java.awt.AWTException is not a subclass of java.lang.RuntimeException
    /// method body
    }
    public int getPrice(int id) {
        if (id == 6)
            throw new java.lang.IndexOutOfBoundsException();
                    // Legal - Reason 2
                    // IndexOutOfBoundsException is derived from
                    // RuntimeException
        else
            ...
    }
    public int getPrice(int id) throws java.lang.IndexOutOfBoundsException {
        // Legal - Reason 1
        // IndexOutOfBoundsException is derived from
        //RuntimeException
        if (id == 6)
            throw new java.lang.ArrayIndexOutOfBoundsException();
                // Legal - Reason 3
                //     ArrayIndexOutOfBoundsException is derived from
                //IndexOutOfBoundsException
        ...
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第13章 线 程

本章内容：

13.1 什么是线程
13.2 为什么使用线程
13.3 如何使类可线程化
13.4 巧妙的线程竞赛
13.5 理解 GreatRace
13.6 线程处理
13.7 测试巧妙的线程竞赛
13.8 改变优先级
13.9 关于线程的优先级、Netscape 和 Windows
13.10 同步
13.11 哲学家进餐问题
13.12 改变线程的运行状态
13.13 获取正在运行的线程的数量
13.14 查找所有正在运行的线程
13.15 超级属性

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

13.1 什么是线程

Java 的一大特性就是内置的线程支持。多线程允许用户可以同时做多件事情。如果一个人在同一个时间里只允许活动一支胳膊或者一条腿，那么这个人肯定觉得受限制了。多线程就是计算机解决这类问题的方案。本章将介绍如何在 Java 程序里使用线程。

就以公司为例。大部分公司都至少有三个独立的部门：管理部、财务部、生产部或销售部。为了使公司有效地运营，三个部门需要同时运作。如果财务部门停止了工作，那么公司就会倒闭；如果管理部门运作不力，公司就会解体；如果生产部门不干活，那么公司就无法赚钱。

许多软件程序都是在与公司一样的条件下运行的。在公司里，可以把任务分给不同的人去完成。每个人去执行自己领受的任务。在软件里，通常用户只有一个处理器，这一个处理器必须接受所有的任务。为了管理处理器，提出了“多任务”的概念，实际上在同一时间里处理器还是只做一件事情，但是通过任务之间的快速切换让人们感到处理器正在同时做多件事情。幸运地是新式计算机的工作速度比人快得多，人们很难看清楚它在某一个时刻做的具体动作。

现在让我们进一步看我们的公司。你是否注意到财务部的人实际上也不只干一件事情。例如影印表格，计算需要卖出多少商品才能占领商品市场，合计帐簿和核实帐单的支付。

用计算机术语讲，就叫做多线程。照这样考虑：每个程序都被交给一个专人去执行一组任务，这个程序就叫做进程。这个人再把他(或她)的时间进一步分成线程。

13.2 为什么使用线程

也许读者会对自己说：“为什么我要关心计算机是如何工作的？只要它能够运行我的程序不就可以了吗？”理解多线程是很重要的，因为 Java 比起其它编程语言最大的优势是它的内置的、本地的对线程的支持。通过使用线程，用户可以避免在执行任务和查看结果之间的等待。借助这个优势，用户可以将打印工作置于后台，而在前台继续输入文档或执行某些其它任务。

在 Java 中，线程通常用于使小应用程序进入后台做某些事情，而让浏览器继续做它自己的工作。任何需要同时做多件事情的应用程序都可以考虑使用 Java 来开发。

13.3 如何使类可线程化

使应用程序和类运行在两个线程内有两种途径：

- 扩展 Thread 类
- 使用 Runnable 接口

13.3.1 扩展 Thread 类

可以通过扩展 java.lang.Thread 使得类可以作为线程来运行。这样可以访问所有的线程方法：

```
public class GreatRace extends Thread
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

13.3.2　使用 Runnable 接口

通常，当用户希望某个类能够运行在自己的线程中时，也希望扩展某些其它类的特性。因为 Java 不支持多继承性，所以必须通过 Runnable 接口来实现。事实上 Thread 类本身就使用了 Runnable 接口。Runnable 接口只有一个方法：run()。不论何时创建了一个使用了 Runnale 接口的类，都必须在类中编写 run()方法来覆盖接口中的 run()方法。在类里的 run()方法中定义了线程具体要做的事情：

public class GreatRace extends java.applet.Applet implements Runnable

13.4　巧妙的线程竞赛

前面介绍了如何使用 Runnable 接口使类可线程化，现在让我们看一下线程示例。两个类的源代码如程序清单 13.1 和 13.2 所示。

- CreatRace．在 Thteader 类中增加了几个项目。
- Thtreader．在自己的线程中运行并跟踪结束行。

```
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.Frame;
import Threader;

public class GreatRace extends java.applet.Applet implements Runnable{
  Threader theRacers[];
  static int racerCount = 3;
  Thread     theThreads[];
  Thread     thisThread;
  static boolean inApplet = true;
Listing 13.1 Continued
  int      numberofThreadsAtStart;

  public void init(){
    //we will use this later to see if all our Threads have died.
    numberofThreadsAtStart = Thread.activeCount();

    //Specify the layout. We will be adding all of the racers one on top
    //of the other.

    setLayout(new GridLayout(racerCount,1));

    //Specify the number of racers in this race, and make the arrays for the
    //Threaders and the actual threads the proper size.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    theRacers = new Threader [racerCount];
    theThreads = new Thread[racerCount];

    //Create a new Thread for each racer, and add it to the panel.
    for (int x = 0;x < racerCount;x ++ ){
      theRacers[x] = new Threader ("Racer #" + x);
      theRacers[x].setSize(getSize().width,getSize().height/racerCount);
      add (theRacers[x]);
      theThreads[x] = new Thread(theRacers[x]);

    }
  }

  public void start(){
    //Start all of the racing threads
    for (int x = 0;x < racerCount;x ++ )
      theThreads[x].start();

    //Create a thread of our own. We will use this to monitor the state of
    //the racers and determine when we should quit altogether.
    thisThread= new Thread (this);
    thisThread.start();
  }

  public void stop(){
    for (int x = 0;x < theRacers.length;x ++ ){
      theRacers[x].stop();
    }
  }

  public void run(){
    //Loop around until all of the racers have finished the race.
    while(Thread.activeCount() > numberofThreadsAtStart + 2){
      try{
        thisThread.sleep(100);
      } catch (InterruptedException e){
        System.out.println("thisThread was interrupted");
      }
    }

    //Once the race is done, end the program.
    if (inApplet){
      stop();
      destroy();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }
    else
      System.exit(0);
  }

  public static void main (String argv[]){
    inApplet = false;

    //Check to see if the number of racers has been specified on the command
    //line.
    if (argv.length > 0)
      racerCount = Integer.parseInt(argv[0]);

    //Create a new frame and place the race in it.
    Frame theFrame = new Frame("The Great Thread Race");
    GreatRace theRace = new GreatRace();
    theFrame.setSize(400,200);
    theFrame.add ("Center",theRace);
    theFrame.show();
    theRace.init();
    theFrame.pack();
    theRace.start();
  }

}//end class GreatRace.
```

```
import java.awt.Graphics;
import java.awt.Color;

public class Threader extends java.awt.Canvas implements Runnable {
  int myPosition = 0;
  String myName;
  int numberofSteps = 600;
  boolean keepRunning = true;

  //Constructor for a Threader. We need to know our name when we
  //create the Threader.
  public Threader (String inName){
    myName = new String (inName);
  }

  public synchronized void paint(Graphics g){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        //Draw a line for the 'racing line'.
        g.setColor (Color.black);
        g.drawLine (0,getSize().height/2,getSize().width,getSize().height/2);

Listing 13.2 Continued
        //Draw the round racer.
        g.setColor (Color.yellow);
        g.fillOval((myPosition * getSize().width/umberofSteps),0,15,getSize().height);
    }

    public void stop(){
      keepRunning = false;
    }

    public void run(){
      //Loop until we have finished the race.
      while ((myPosition < numberofSteps) && keepRunning){
        //Move ahead one position.
        myPosition++;
        repaint();

        //Put ourselves to sleep so the paint thread can get around to painting.
        try{
          Thread.currentThread().sleep(10);
        }catch (Exception e){System.out.println("Exception on sleep");}
      }
      System.out.println("Threader:" + myName + "has finished the race");
    }

}//end class Threader.
```

13.5 理解 GreatRace

只要用户读了本书前面的内容，那么 Threader.java 和 GreatRace.java 的代码就不难理解。让我们看一下代码的关键部分，即对实际的线程的处理。首先看一下 GreatRace 类的 init()方法中的 for 循环，如程序清单 13.3 所示。

```
for (int x = 0; x < racerCount; x++ ){
    theRacers[x] = new Threader ("Racer #" + x);
    theRacers[x].resize(size().width,size().height/racerCount);
    add (theRacers[x]);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        theThreads[x] = new Thread(theRacers[x]);
    }
```

在 for 循环中，第一件要做的事情就是建立 Threaer 类的实例。正如用户在程序清单 13.2 中看到的那样，Threader 普通地使用了 Runnable 接口的类。当 Threader 的实例被创建以后，该实例即被加进了面板，并且有 Threader 参数的新线程被创建。不要将 Threader 类和 Thread 的类搞混。

警告：新的 Thread 类只能使用扩展 Thread 类的对象和使用 Runnable 接口的对象来调用。不论怎样调用，对象都必须拥有 run()方法。当第一次创建线程时，run()方法并没有被调用。调用是发生在线程被启动之后。

第二个重要的代码集包含在 start()方法中，现在看一下 GreatRace.java 中的 start()方法，如程序清单 13.4 所示。

```
public void start(){
  //Start all of the racing threads.
  for (int x = 0;x < racerCount;x ++ )
    // start() will call the run() method.
    theThreads[x].start();

//Create a thread of our own. We will use this to monitor the state of
//the racers and determine when we should quit altogether.
  thisThread = new Thread (this);
  thisThread.start();
}
```

第一件事情是启动 init()方法中的所有线程。当线程被启动时，系统通过 Runnable 接口调用 run()方法。在这里调用是在 init()方法中传递给构造器的 Threader 对象的 run()方法。

请注意，当竞赛者启动以后，就有一个供实际的小应用程序使用的线程被创建，这个线程将被用于监视所有的线程正在做的事情。当竞赛结束(即所有的线程都已经死亡不再活动)时，用户也就结束了程序。

最后看一下 Threader 类的中最后一个重要的代码集—Threader 类的 run()方法，如程序清单 13.5 所示。

```
public void run(){
  //Loop until we have finished the race.
  while ((myPosition < numberofSteps) && keepRunning){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        //Move ahead one position.
        myPosition ++ ;
        repaint();

        //Put ourselves to sleep so the paint thread can get around to painting.
        try{
          Thread.currentThread().sleep(10);
          }catch (Exception e){System.out.println("Exception on sleep");}
        }
      System.out.println("Threader:" + myName + "has finished the race");
    }
```

注意 while 循环很长。run()方法只在线程启动时被调用一次。如果用户打算做许多重复工作,那么就需要停留在 run()方法的限制中。事实上把 run()方法当作其它语言中 main()方法来理解不是什么坏主意。

再接着看代码中的下几行,注意在每个循环中都有一个线程休眠(Thread.currentThread().sleep(10))。这是一个重要的任务。用户应该不时地让线程休眠以免其它线程饥饿(得不到对处理器的控制,译者注)。

在 Windows 系统下用不着编写这种代码。就像将在"关于线程优先级,Netscape 和 Windows"部分中讨论的那样在有优先级存在的前提下 Windows 的行为并没有像它应该地那样。这种休眠的做法不是好主意,它也不利于移植。在 UNIX 机器上看上去就像小应用程序已经被挂起了,在 Macintosh 上也是如此。另外必须给屏幕绘制程序设置适当的优先级,还有一些其它的原因要求系统有喘息之机,不能一直围着用户的线程转。

13.6 线程处理

为了更好地理解让线程休眠的重要性,首先应该理解计算机是如何实际执行线程的。计算机是如何把多个线程处理得让用户看起来像是同时在做多件事情一样的?答案就在于任务切换的本质。

在计算机内部有一个周期时钟。比如说这个时钟每毫秒(千分之一秒)内滴嗒一次(实际的计算机时钟比这快)。现在,计算机每毫秒看一下进程表。在这个表中有指向当前正在运行的进程(和线程)的指针。然后计算机检查是否有哪个线程需要运行,如果没有就马上回到前面正在运行的线程。这个过程显示在如图 13.1 所示的时间轴上。

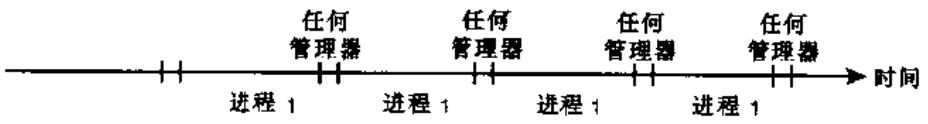


图 13.1 当只有一个进程在运行时,任务管理就一直跟踪这个进程

如果任务管理器发现进程表中有多个等待运行的线程,如果这些线程具有着相同的优先级,那么系统就轮流地执行它们,图 13.2 显示了其激活过程。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

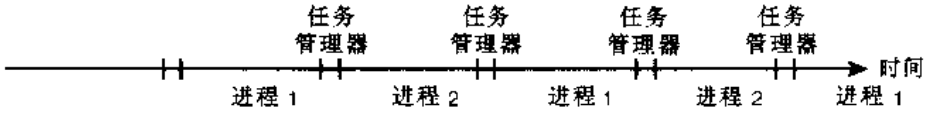


图 13.2　当有两个具有相同优先级的进程都在运行时任务管理器负责切换这两个进程

如果任务管理器发现这是两个优先级不同的线程，比如说进程 2 的优先级低于进程 1。在这种情况下，任务管理器只运行优先级较高的进程。这一段时间轴如图 13.3 所示。

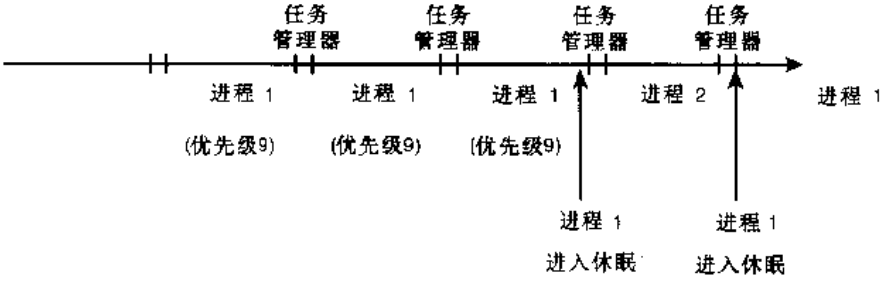


图 13.3　任务管理器总是返回到优先级较高的进程直到它进入休眠状态

13.7　测试巧妙的线程竞赛

接下来我们编译 GreatRace，然后通过输入下列命令来运行它，

```
java GreatRace
```

结果如图 13.4 所示。用户也可以通过在浏览器中打开 index.html 文件来运行它。



图 13.4　作为应用程序运行的 GreatRace

用户可以看到三个椭圆穿过屏幕。它们几乎以相同的速度前进，它们真是不同的线程吗？用户可以运行有任意多个竞赛者的 GreatRace，可以输入：

```
java GreatRace 5
```

所有的竞赛选手几乎同时穿过屏幕(见图 13.5)。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

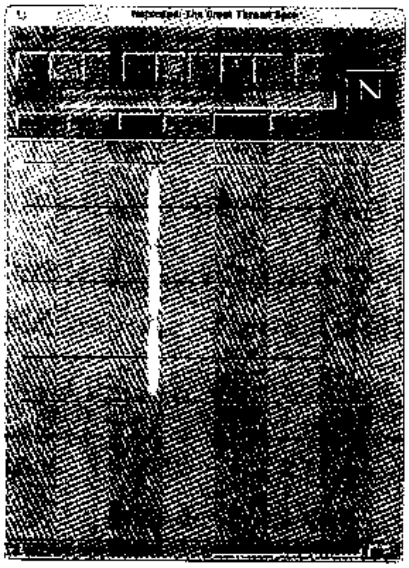


图 13.5 作为小应用程序运行的 GreatRace

如果用户多次运行这个竞赛，能够发现这个竞赛是非常公平的，每个参赛选手获胜的机会都是均等的，如果在 Netscape 下显示控制台或查看运行 java GreatRace 的窗口，用户可以实际地看到参赛选手到达终点的顺序，如图 13.6 所示。

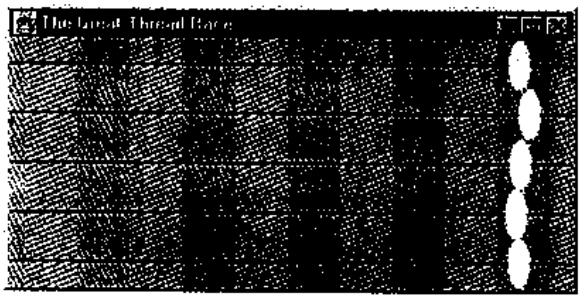


图 13.6 显示 GreatRace 的窗口和运行 java GreatRace 的 DOS 窗口

13.8 改变优先级

在 java.lang.Thread 类中有两个处理线程优先级的方法：

- SetPriority(int) 用于为线程设置新的优先级。
- GetPriority 用于获取当前线程的优先级。

让我们看一下当我们告诉计算机某些参赛选手的优先级别不同时将会发生什么情况。

通过在 for 循环中增加下列代码来修改 GreatRace.java 的 init()方法：

```
theThreads[x].setPriority (Thread.MIN_PRIORITY + x);
```

那么现在的 for 循环就变成了如程序清单 13.6 所示的样子：

程序清单 13.6 init()方法的新 for 循环

```
//Create a new Thread for each racer, and add it to the panel.
for (int x = 0; x < racerCount; x++ ){
    theRacers[x] = new Threader ("Racer #" + x);
    theRacers[x].setSize(getSize().width, getSize().height/racerCount);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        add (theRacers[x]);
        theThreads[x] = new Thread(theRacers[x]);
        theThreads[x].setPriority(Thread.MIN_PRIORITY + x);
    }
```

现在重新编译 GreatRace,然后再运行它,结果如图 13.7 所示。

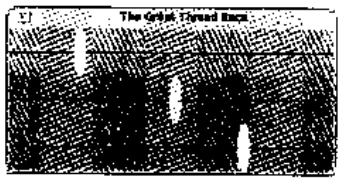


图 13.7　新的正在运行的 GreatRace

通过修改参赛选手的优先级,比赛结果突然变成全是底下的选手获胜。为什么?只要具有最高优先级的线程不休眠,那么它就一直拥有对处理器的使用权。这就意味着每隔 10 毫秒,底部的选手就向前移动一些,此时其它选手停下来。只有当底部选手决定休眠时其它选手才有机会向前追赶。这与兔子和乌龟寓言中的兔子不同,最高优先级的线程总是在 10 毫秒时间内被唤醒,并且马上就可以超出其它选手并直奔终点。只有该选手到达终点,另一个选手才能成为具有最高优先级线程的选手,每 10 毫秒移动一次,把其它选手抛在后面。

注意:线程的优先级是使用线程的 setPriority(int)方法修改的。注意,并不是只给它一个数就可以,优先级的设定要使用线程中的 MAX_PRIORIITY 变量。这是很重要的步骤。MIN_PRIORITYT 和 MAX_PRIORITY 的设置对于不同的机器所有差别。目前各机器的 MIN_PRIORITY 为 1,而 MAX_PRIORITY 为 10。不能超过这个值。如果超过了这个值,将引发 IllegalArgumentException 异常。

13.9　关于线程的优先级、Netscape 和 Windows

如果在 Windows 操作系统上运行更新过的 GreatRace 版本,将看到如图 13.8 所示的结果。用户一定会对结果与图 13.7 不同而感到惊讶。后面两个选手位置十分接近,直到第一个选手抵达终点。

在 Windows 中使用 Netscape 运行新的 GreatRace,如 13.9 所示,这时用户可能会因最后一位选手没有获胜而感到奇怪。

这种差异是因为线程在 Windows 中不太受优先级控制,不像 UNIX 系统或 Macintosh 机器中的线程。事实上,具有近似相同的优先级的线程在 Netscape 的 Windows 版本中是被作为相同的优先级来处理的。这就是在 Netscape 中的最后两个选手看起来具有相同的获胜机会的原因。为了使最后一个选手总赢,必须增大优先级差别。将 GreatRace 的 init()方法中优先级设定语句设成如下所示的样子:

```
the Threads[x].setPriority(Thread.MIN_PRIORITY + x * 2);
```

现在再在 Windows 95 中运行竞赛程序,最后的选手又会胜一截,如图 13.10 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

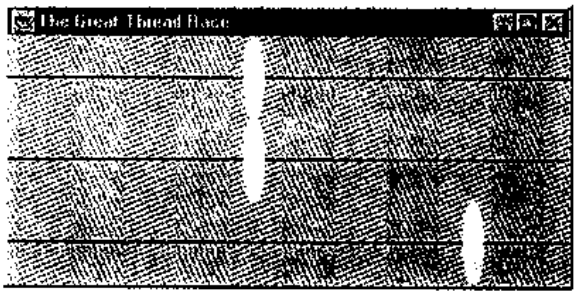


图 13.8 运行在 Windows 95 上的新 GreatRace

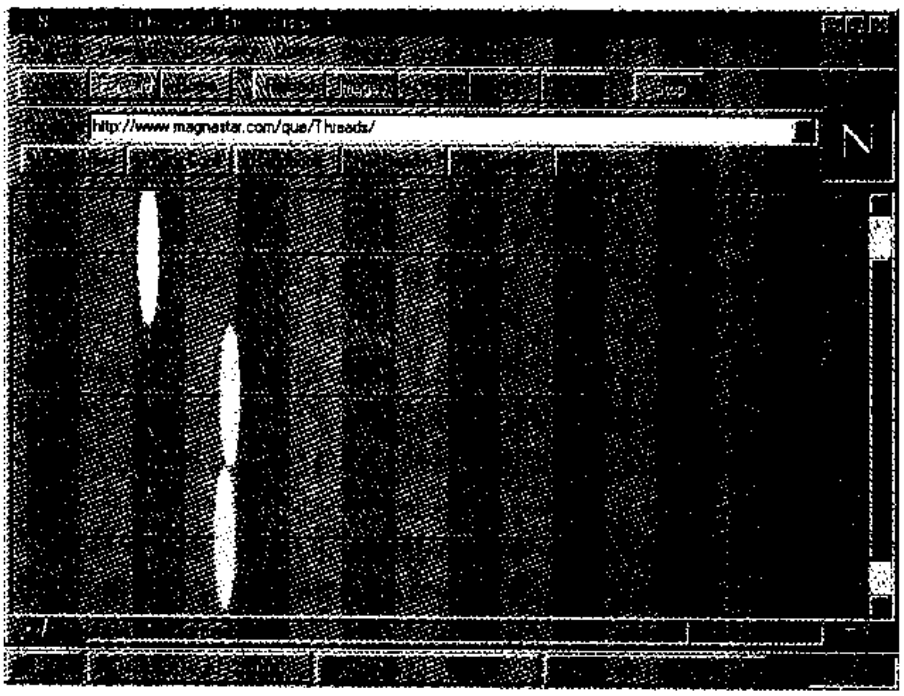


图 13.9 在 Windows 95 中当作小应用程序运行的新 GreatRace

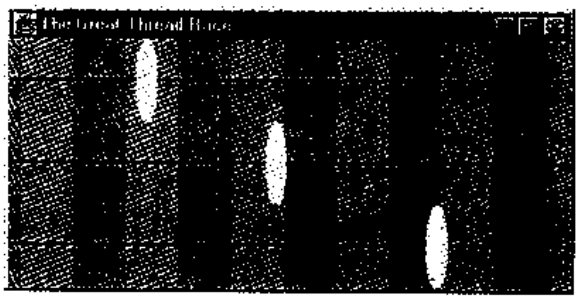


图 13.10 在 Windows 95 下增大了优先级的 GreatRace

如果再在 Netscape 下运行,最后一个选手又成了赢家,但是领先不多,如图 13.11 所示。

这个差别对于实现是很重要的。如果依靠线程的优先级,必须把应用程序分别在 Windows 和 Macintosh 或 UNIX 机器上测试。如果用户没有奢侈的 UNIX 机器或 Macintosh 机器,那么当作应用程序来运行比当作小应用程序运行对优先级处理更接近合理一些,就像在图 13.10 和 13.11 中看到的那样。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

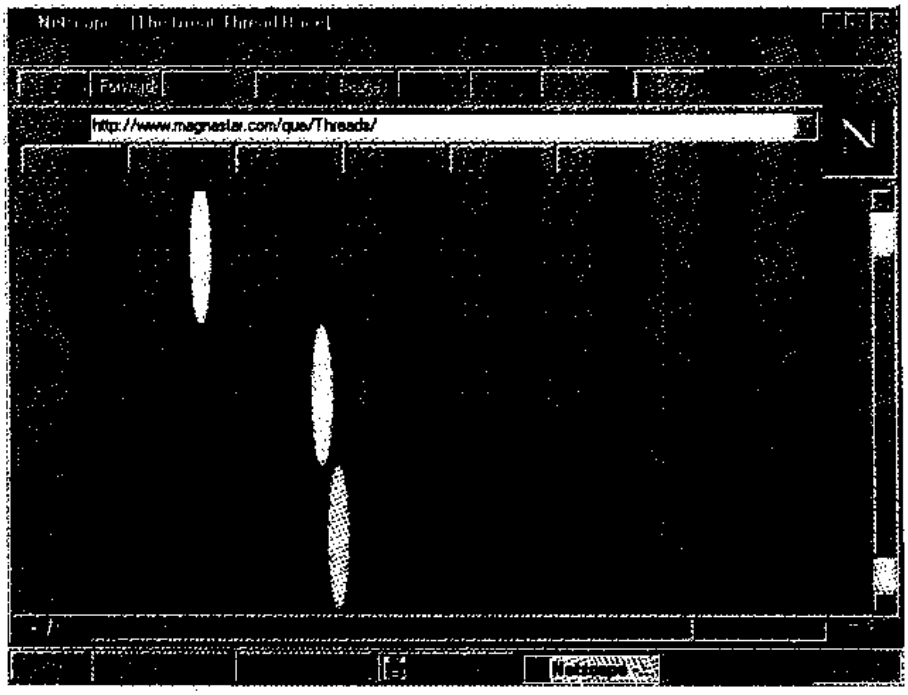


图 13.11　增加优先级后再在 Wondows 95 下作为小应用程序来运行

警告：如果在 Wondows 95 机器上不为线程设置定期的休眠，那么这种线程优先级差别将使得它变得危险。绘制屏幕线程是低优先级的线程，在 Windows 下可以有机会得到对处理器的控制使用，但它也仅仅是像一下参赛选手一样等待这个机会。但是在 Macintosh 机器或 UNIX 机器上不是这样。

13.10　同步

在处理多线程时，用户是否考虑过这个问题：当两个或多个线程同时访问同一个变量，并且至少有一个线程需要修改这个变量，这时会发生什么情况呢？如果允许这些线程这样做，可能会引起混乱。例如，当一个线程读 Joe Smith 的记录时，另一个线程正试图改变他的工资（比如要增加 50 美分）。问题是这种小小的改变会引起正在读文件的线程的误解，因为别人的修改对于它来说是随机的，它有可能认为 Joe 增加了 500 美元薪水。这对于 Joe 本人当然是一件大好事，但对于公司则可能是不好的事情，对程序员来说更糟糕的是他可能会为此而失去工作。如何解决这个问题呢？

要做的第一件事情是把要修改数据的方法和要读可能被修改的数据的方法声明为同步的。Java 的关键字 synchronized 告诉系统为某个特殊的方法加锁。大部分情况下，一个线程里可以有多个同步方法。程序清单 13.7 显示的例子中包含有两个同步方法。

```
public synchronized void setVar(int){
  myVar = x;
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public synchronized int getVar (){
  return myVar;
}
```

现在当程序运行到 setVar()时，Java 虚拟机设置一个条件锁，不许其它线程进入 synchronized 方法，包括 getVar()方法，直到 setVar()方法结束。因为不允许其它线程进入 getVar()方法，所以就没有线程在 setVar()方法正在写的过程中得到不正确的信息。

不要把所有的方法都设置成同步的，如果都设置成同步的，程序就不再是多线程的程序了。因为其它线程要等到锁被打开以后才能继续运行，所以此时只有一个线程处于活动状态。即使是只有两个方法被声明成了 synchronized，那么当某个线程启动了一个 synchronized 方法，直到遇到必须由其它方法来设置的条件时才停止运行，这时其它的线程不就必须进入 synchronized 方法了吗？这个答案就和哲学家进餐问题的答案差不多。

13.11 哲学家进餐问题

什么是哲学家进餐问题？在这里我们不做详细讨论，只做一点概略地介绍。

有五位哲学家围坐在餐桌边上等待进餐。每人面前只有一根筷子，总共五只，当大家都要吃饭时该怎么办？人们在进食食品时一般是需要使用两根筷子的。但眼下却没有那么多筷子。对于普通人遇到这种情况会先让两个先吃，其它三个人等着来解决这个问题。但是每个哲学家都认为：我若不拿一根筷子，怎么可能拿到第二根筷子呢？所以大家就拿着一根筷子等待第二根筷子。这样做的结果是大家都挨饿，因为谁也拿不到两根筷子。（哲学家太忙，想不起来派人到厨房再拿几根筷子来；但是这不是本问题的解决方案。）

有许多方法可以解决这个古老的问题。本人不想在这里解答这个问题。但是认清事情的结果是很重要的。如果把方法设成 synchronized，那么当其它线程设置了某些条件时该方法才停止运行，退出并把筷子放回到桌上。如果不这样做，将会出现饥饿的等待。哲学家不会将筷子放回桌上，而是等待不可能发生的事情发生，因为他的同伴也没有器皿用来开吃。

13.12 改变线程的运行状态

线程有几种可能的状态。让我们看一下如何改变这些状态和改变状态后的结果。改变线程的方法有以下几个：

- start ( )
- yield ( )
- destroy ( )
- sleep(long), sleep(long ,int)
- * stop( )
- * resume ( )
- * suspend ( )

打 * 号的方法是不提倡的，所以一般不使用它们。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

start()和 stop()对于线程来说是简单的操作。start()告诉线程启动有关的 Runnable 对象的 run()方法。

stop()告诉线程停止运行。它实际上是在线程中产生一个 ThreadDeath 对象。几乎在每种情况下,用户都不必捕捉这个对象。只有当在停止之前需要请除某些特殊的东西时才需要考虑这样做。

警告:在 JDK 1.2 中不提倡使用 stop(),suspend(),和 resume()方法。这是因为它们必然地会导致死锁,就像前面讲到的哲学家那样。放弃这些方法,通过在 run()方法中使用条件一样可以达到这些目的。

警告:如果捕获到 ThreadDeath 对象,必须激活相应的异常处理。如果不这样做,线程将不会真正停止,因为错误处理器不会注意它,所以就不会自动地处理它。

前面已经简单地介绍了一下 sleep()方法,介绍了何时将 GreatRace 的 Threadable 置为休眠状态。将线程置成休眠状态等于告诉虚拟机:"我要停下来休息一下,请过一会儿叫醒我"。通过将线程设置成休眠状态,可以使低优先级的线程有机会得到处理器的使用权。这对于那些做不太重要但仍需要做的低优先级工作来说能够有机会运行。如果没有线程休眠,那么有些线程就要面临饥饿的危胁了。

Sleep( ) 方法有两种形式,第一种形式是 sleep(long),sleep (long)告诉解释器,该线程要休眠较长的时间:

```
this Thread.sleep(100) ;
```

一毫秒对于人类来说只是一瞬间,但是对于计算机来说则是很长的一段时间。对 486/33 机器而言,处理器在这个时间里可以处理 25,000 条指令。而对于高级的工作站,在一毫秒时间里则可以执行数十万条指令。

由于这个原因,所以设置了另一种方法 sleep (long,int)。使用这种方法可以把线程休眠时间设置为×××毫秒和×××纳秒。

```
thisThread.sleep (99,250);
```

supend ()方法和 resume()方法可以用于将线程设置为休眠直至某个其它事件上发生。例如当用户要开始大量数学计算时,比如要求出第一百万个素数,在此期间不希望其它线程占用处理器,直到求出这个素数为止。(如果用户真要求出第一百万个素数,我建议用户不要使用 Java,而使用其它的语言,比如计算大王 fortran 语言,并且必须有一台大型计算机)。

另外,在 JDK 1.2 中也不提倡使用 suspend()和 resume()。用户应该用包含 wait()方法和 notify()方法的线程来监视这种线程。

yield ()方法与 suspend()有点不同。yield()方法更接近 sleep()方法。使用 yield()方法可以告诉解释器摆脱其它线程的方式,当这些线程被执行完毕时,再执行 yield()。yield()方法不要求在其它线程停止并进入休眠状态或退出时借助于 resume()方法来启动。

最后一个改变线程的运行状态的方法是 destroy()。一般情况下不要使用 destroy()方法。destroy 方法不对线程执行任何清理工作;只是撤销线程。因为它与简单的关闭程序进程一样,所以只能把 destroy()作为最后的手段来使用。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

13.13 获得正在运行的线程的数量

java.lang.Thread 类只有一种计算正在运行的线程的数量的方法:activeCount()方法。

Thread.activeCount ( )返回一个整数,这个整数就是当前 ThreadGroup 中正在运行的线程的数量。当所有的线程都结束运行时在 GreatRace 中使用这个方法。注意在 init ()方法中当用户启动自己的程序时应当检查正在运行的线程的数量。在 run()方法中,将这个数加 2 后与当前正在运行的进程的数量进行的比较,看是否有哪位选手已经结束了竞赛:

While (Thread.activeCount ( ) > numberofThreadsAtStart + 2){

注意:为什么要加 2 呢? 因为这两个线程是在竞赛之前没有的附加线程。第一个线程在运行 GreatRate (this Thread)之前就有了,它实际是由 GreatRace 的 main 运行的。另一个线程是 Screen _ Updater 线程,它在 init()方法启动时尚未启动。直到需要该线程工作时才启动该线程。

提示:就像其它程序一样,可以使用多种办法来知道所有选手是否到达终点。可以使用线程传送 PipedInputStream 和 PipedOutputStream 信息,也可以查看线程是否处于激活状态。

13.14 查找所有正在运行的线程

有时需要查看所有正在运行的线程。例如,有时需要知道在 GreatRace 的 main()中有几个需要累计的线程。java.lang.Thread 类中有几个方法可以帮助用户显示这个信息:

- enumerate (Thread[ ])
- getName ( )
- setName (String)

enumerate (Thread[ ])用于获取当前 ThreadGroup 中正在运行的线程的列表。getName ( )用于获取赋给线程的名字,与之相对的 setName (String)用于设置这个名字。在缺省情况下,如果用户不向线程构造器传递线程的名字,系统将使用 Thread - x 来为线程命名,其中 x 是一个数字,每个线程都使用不同的数字。

经过对 GreatRace 做小小的修改,可以显示所有正在运行的线程。程序清单 13.8 对 run ()方法做了修改。

```
public void run(){
  Thread allThreads[]
  //Loop around until all of the racers have finished the race.
  while(Thread.activeCount() > 1){
    try{
```

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
        //Create a Thread array for allThreads.
        allThreads = new Thread[Thread.activeCount()];
        //Obtain a link to all of the current Threads.
        Thread.enumerate(allThreads);
        //Display the name of all the Threads.
        System.out.println("* * * * * * New List * * * * * *");
        for(int x = 0;x < allThreads.length;x ++ )
          System.out.println("Thread:" + allThreads[x].getName()
➡ + ":" + allThreads[x].getPriority() + ":" + allThreads[x].isDaemon());
        thisThread.sleep(1000);
      }catch(InterruptedException e){
        System.out.println("thisThread was interrupted");
      }

  //Once the race is done,end the program.
  if(inApplet){

    destroy();
    }
  else
    System.exit(0);
  }
```

While 循环中的头几行是新加的，这几行使用刚刚讲过的 enumerate 方法创建了一个线程数组，并且把每个线程的名字写到 System.out 。

现在重新编译该程序并运行它。在 Netscape 中注意选择 Options 打开 Java 控制台。结果如图 13.12 所示。

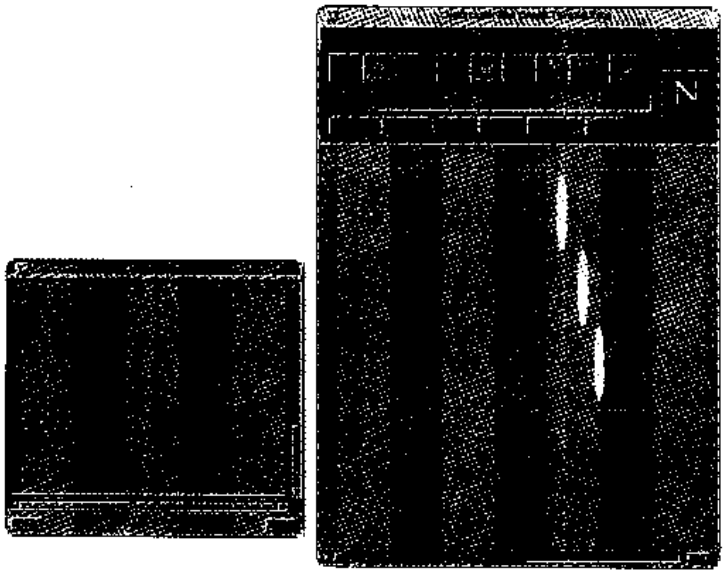
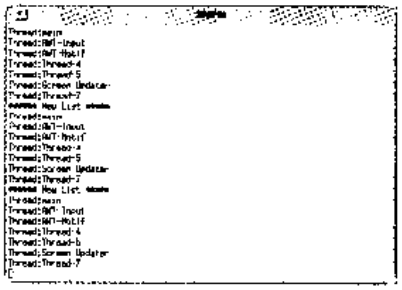


图 13.12　当运行 Great Race 时的 Netscape 控制台

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

随着每个竞赛进程的结束,用户可以看到活动的线程在逐个递减。事实上,用户可以多选几个选手(3 个以上)参加竞赛(见图 13.3),例如:

java GreatRace 5



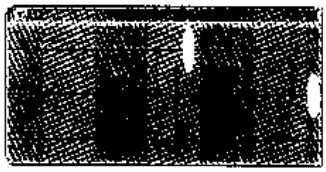


图 13.13 有 5 个选手参赛的 GreatRace

13.15 超级属性

线程可以分为两种类型:用户线程和超级线程。

那么什么是超级线程呢?在英文中超级线程用的是 daemon 一词。在韦伯词典中 daemon 一词的解释是:"超自然的存在或力量,不是邪恶。"在某种意义上讲,韦伯词典的解释,讲出了超级线程的含义。虽然超级线程不是超自然的,当然也肯定不是邪恶的,但是超级线程也不是普通的线程。当启动超级线程时,用不着管它何时停止。当该线程到达指定任务的终点时,它就会自动停下来并将自己的状态改为非激活状态,这一点很像用户线程。

超级线程与用户线程最重要的差别是超级线程可以一直运行下去。如果解释器发现程序只有超级线程在运行,那么就可以把程序退出来,解释器并不担心超级线程是否结束。这是很有用途的,因为它允许用户启动监视线程;当没有其它东西可运行时监视线程就可以自动停止工作。

这种技术在图形化的 Java 应用程序中是受限制的,因为在缺省情况下,一些基本的线程不能被设置成超级。它们包括:

- AWT - Input
- Main
- AWT - Motif
- Screcn _ Updater

不幸的是,这意味着所有使用了 AWT 类的应用程序都不能使用超级线程,目的是防止应用程序退出。

java.lang.Thread 类有两个方法负责设置线程的超级属性:

- IsDaemon()
- SetDaemon(boolean)

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

第一方法，isDaemon()用于测试某个指定线程的状态。当某个对象作为一个线程来运行时，该方法可以鉴别出它是超级线程还是普通线程。如果线程是超级线程。IsDemon()返回true，否则返回false。

第二个方法setDaemon(boolean)用于修改线程的超级状态。为了使线程成为超级线程，可以把参数值设成true。要想把它改回用户线程，可以把init()方法改成如程序清单13.9所示的样子。

```
for(int x = 0;x < racerCount;x ++ ){
    theRacers[x] = new Threader("racer #" + x)
    theRacers[x].resize(size().width,size().height/racerCount);
    add(theRacers[x])
    theThreads[x] = new Thread(theRacers[x]);
    theThreads[x].setDaemon(true);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 14 章　编写小应用程序

本章内容：

14.1　Java 的孩子
14.2　小应用程序和 HTML 语言
14.3　在 HTML 页中包含 Java 小应用程序
14.4　使用 Java 插件
14.5　开始开发 Java 小应用程序
14.6　探究小应用程序的生命周期
14.7　使用控件的小应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

14.1　Java 的孩子

最先出现的是 FTP,然后是 Telnet;几年之后,Telnet 诞生出 Web(网页)。在 CGI 出现以前网页是静态的,缺乏活力的,但是 CGI 要求确认按钮和下载整个网页;这时候业界人士都认为这种方法不好,然后有一些人从废墟中找到了 Oak,把它当作不死鸟一样恢复和完善。并用它建造了动态的具有客户/服务器支持的网页。它们把 Oak 重新命名为 Java,而 Oak 的孩子,他们就称之为小应用程序。这时的世界再也没人说它不好了。

如果用户没有使用过 Java,看到这里你可能会十分迫切地要学习编写 Java 小应用程序。小应用程序就是那些在国际互联网上可以看到的那些 Java 程序。小应用程序在已经是动态的 Java 语言之上提供了更具吸引力的层次,它扩展了传统的编程结构和方法。当用户编写小应用程序时,小应用程序不能单独运行,但是可以被包含在标准 HTML 页中运行。既然用户已经学习了 Java 语言,那么肯定会为将要开始编写小应用程序而兴奋不已,因为这些小应用程序可以通过国际互联网看到。在本章中将介绍如何编写 Java 小应用程序。

14.2　小应用程序和 HTML 语言

既然有编写 Java 小应用程序的兴趣,那么用户可能已经熟悉了用 HTML(超文本标识语言)创建网页了。如果不是这样,用户可以通过阅读一本 HTML 的书,比如 Que 出版的《HTML 4 使用大全》来了解标识语言是如何工作的。

正如用户所知道的那样,Java 可以用于编写两种类型的程序:小应用程序和独立运行的应用程序。小应用程序必须包含在网页中,就像图像和文本行一样。当支持 Java 的网页浏览器加载一个包含对小应用程序引用的 HTML 文档时,小应用程序同时被加载并运行。(参考第 1 章"Java 能做什么"可以了解更多的信息。)

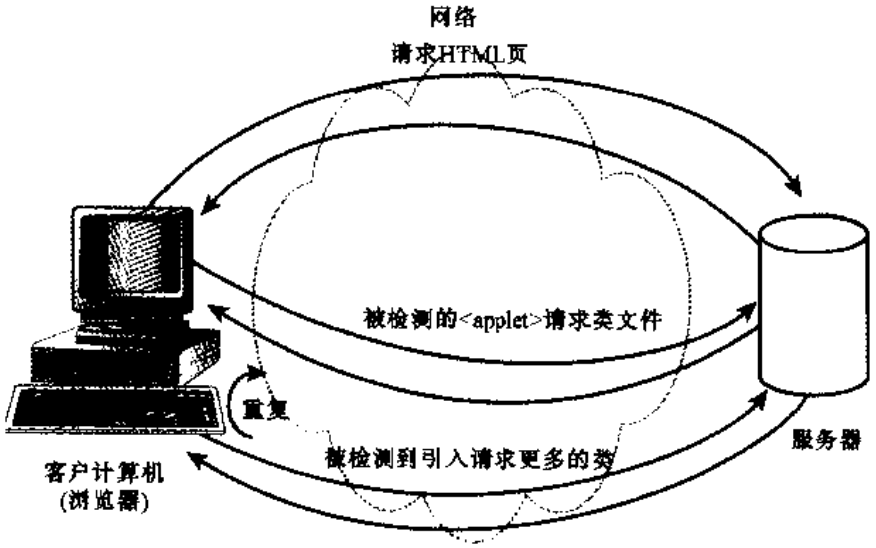


图 14.1　字节码验证将继续返回到服务器查找类文件,
直到所有的小应用程序的代码被下载完毕为止

让我们快速地回顾一下小应用程序的代码是如何在计算机上运行的。当浏览器在 HTML 文件中发现 < APPLET > 标识时,浏览器将从服务器上获取该类文件。字节码验证器

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

将验证这个类是否合法。假设这个类是合法的，验证器就开始处理类文件。当虚拟机发现引入语句时，字节码验证器就继续返回到服务器查找更多的类文件，直到将小应用程序用到的代码全部下载为止。图 14.1 描述了这个循环过程。

14.3 在 HTML 页中包含 Java 小应用程序

如果用户学习本章的主要目标是能够在网页上显示“Java Compliant”标识语的话，读完这一部分就够了。获得 Java 小应用程序最简单的手段是拿一个已经建好的或者让别人给写一个。如果没有时间阅读本书后面的内容并自己编程的话这可能是一个可取的做法。现在看一下如何把 MagnaStar 公司的名为 Muncher 的小应用程序包含在网页上。

程序清单 14.1 是一个最简单的 HTML 文件，它用于显示 Muncher 小应用程序。

```
<HTML>
<BODY>
<APPLET  CODE="GobLoader.class" HEIGHT=0 WIDTH=0></APPLET>
</BODY>
</HTML>
```

注意第三行的 <APPLET> 标记。<APPLET> 标记指示浏览器用户想在网页上包含一个小应用程序。在某种程度上 <APPLET> 标记类似于 <IMG> 标记。关于 <APPLET> 标记有三个关键的属性需要引起用户的注意：CODE、HEIGHT 和 WIDTH。

注意：像大部分 HTML 标记一样，<APPLET> 标记不是大小写敏感的。下面的三个标记具有同样的效果：

```
<APPLE CODE = "GobLoader.class" height = 0 width=0></APPLET>
<Applet code = "GobLoader.class" HEIGHT = 0 WIDTH=0></Applet>
<apPLEt cOdE = "GobLoader.class" height = 0 width=0></ApPLET>
```

虽然 <APPLET> 标记本身不是大小些敏感的，但是它的属性值可以是大小写敏感的。不能把 GobLoader 输成 gobloader 或 GOBLOADER。

<APPLET> 标记的第一个属性是 CODE 语句。<APPLET> 的 CODE 值与 <IMG> 的 SRC 值类似。在 <APPLET> 中 CODE 值必须被设置成小应用程序的主类文件的名字。在 Muncher 中，有许多的类，但只有一个应该包含在 HTML 文件中，这个类文件就是 GobLoader.class（Muncher 通常叫做 Gobbler，所以名字继续沿用）。一定要注意：包含了错误的类名可能引发意想不到的问题，甚至是灾难。另外要记着为 <APPLET> 标记设置 CODE 值，除非 OBJECT 属性不出现。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意：大部分小应用程序会带有一个所包含的类文件的描述或者一个可以找到这些描述的 HTML 文件的示例。或者可以通过浏览在另一个站点上的 HTML 文档资源中找到该类的名字。

第二个和第三个属性是 HEIGHT 属性和 WIDTH 属性。这两个属性与 < IMG > 标记中的相应属性是基本一致的。只有一点对于小应用程序来说与图像不同。那就是有些小应用程序，比如 Muncher，在网页上并不实际地占用空间。取而代之的是它们会创建自己的窗口。这就意味 HEIGHT 和 WIDTH 属性应该设为 0。另外与图像不同，几乎所有小应用程序都得设置 HEIGHT 属性和 WIDTH 属性。对于图像，如果用户不为它指定高度和宽度，浏览器可以计算出图像的大小。但是小应用程序却不能。

对于 < APPLET > 标记，最后一件需要注意的事是结束的 < /APPLET > 标记。另外，正如你在程序清单 14.2 中所看到的，因为 < APPLET > 标记不像 < IMG > 那样拥有 ALT 属性，所以在 < /APPLET > 标记之前的空格可用于包括替换信息。

14.3.1　包括替换信息

程序清单 14.2 显示了一个更完整的 Muncher 的 HTML 文件(见图 14.2)。

```
< HTML >
< BODY >
< APPLET CODE = "GobLoader.class" HEIGHT = 0 WIDTH = 0 >
Warning: You are not using a Java browser. There is an applet on this
page you cannot see.
If you had a Java - enabled browser you would see something similar to the picture Below < BR >
< IMG SRC = "gobbler.gif" ALT = "Game Picture" >
< /APPLET >
< /BODY >
< /HTML >
```

正如用户可以看到的那样，在 < APPLET > 和 < /APPLET > 标记之间可以包括任意的标准 THML 语句。非 Java 浏览器将忽略 < APPLET > 标记，只读取该标记之间的信息。

< PARAM > 标记　Java 小应用程序除 < APPLET > 以外还有一个标记。这个 HTML 标记是 < PARAM > 。许多小应用程序使用该参数标记说明与小应用程序行为有关的信息。现在看另外一个小应用程序，该小应用程序使用了 < PARAM > 参数，小应用程序 GrayButton 也是来自于 MagnaStar 公司的。它提供了向网页增加交互信息的手段，如图 14.3 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

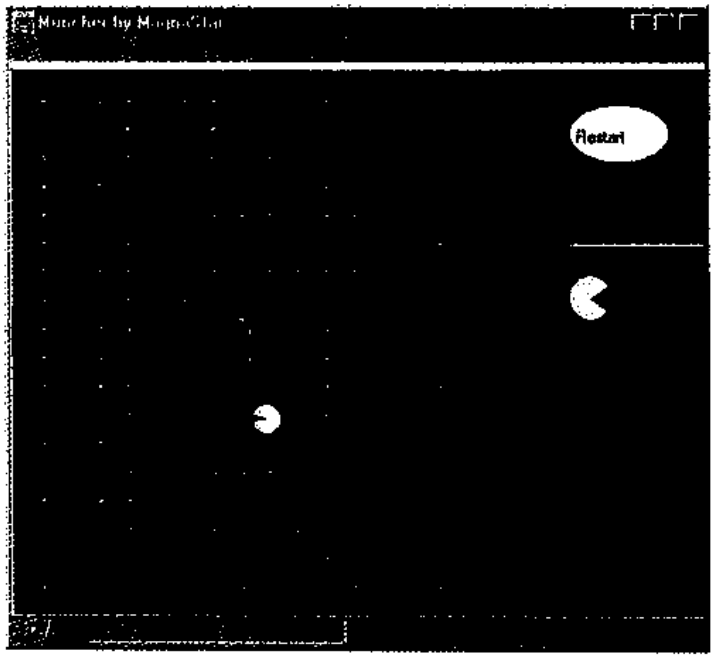


图 14.2 Muncher 是一个共享小应用程序软件，可以从国际互联网获得

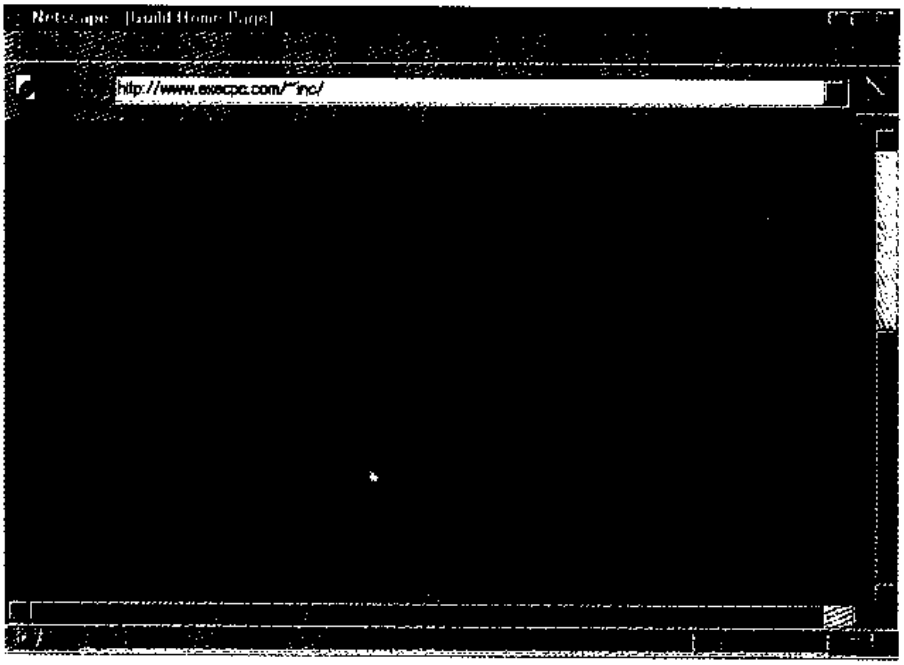


图 14.3 GrayButton 小应用程序用于提供某些受限制的交互

含有 GrayButton 的完整网页代码如程序清单 14.3 所示。

```
<HTML>
<BODY>
<APPLET CODE="gray.class"  WIDTH=300 HEIGHT=300>
<PARAM NAME="graphic" VALUE ="http://www.magnastar.com/NOW.GIF">
<PARAM NAME ="link" VALUE="http://www.magnastar.com/GrayButton/license.html">
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<A HREF="license.html"><IMG SRC="NOW.GIF"></a>
</APPLET>
</BODY>
</HTML>
```

这个示例展示了两件重要的事情。第一个需要注意的事情是第 4 行和第 5 行的 <PARAM> 标记。为了使这个小应用程序得以运行，必须为它指定一个图形供它加载和一个位置供它连接(当单击按钮时)。现在，让我们看一下 <PARAM> 标记的语法。

<PARAM> 标记必须包含在 <APPLET> 标记和 </APPLET> 标记之间，其他位置上的 <PARAM> 标记没有引用点，所以浏览器将忽略它。

通常情况下，<PARAM> 标记有两个自己的属性：NAME 和 VALUE。NAME 属性用于指定用户设置的参数。在 GrayButton 中有两个名字必须设定：graphic 和 link

<PARAM> 标记的第二个属性是 VALUE。VALUE 属性用于指定与 NAME 关联的值。VALUE 不一定是字符串，虽然在 GrayButton 中它们都是字符串，如果小应用程序调用的是数字型数据，那么 VALUE 就是数字。

> **注意**：除 <PARAM> 标记以外，程序清单 14.3 还演示了在 </APPLET> 标记之前的图像连接的使用。这是替代显示的又一个例子。如果浏览器不支持 Java 就会显示这个图形。在 GrayButton 中的显示结果非常好，因为对非 java 浏览器而言损失的仅仅是交互。

14.3.2　附加的 <APPLET> 属性

除了前面提到过的属性以外。用户还可以使用几个属性值进一步定制小应用程序的行为。这些属性如表 14.1 所示。

表 14.1　<APPLET> 标记

属性	值	描述
CODE *	类名	定义扩展 java.applet.Applet 的类文件的名字
HEIGHT*	数字	小应用程序在网页上占据的垂直像素高度
WIDTH*	数字	小应用程序在网页上占据的水平像素宽度
VSPACE	数字	小应用程序和剩下的 HTML 语句之间的垂直距离。与 <IMG> 标记中的 Vspace 的值含义相同
HSPACE	数字	小应用程序和剩下的 HTML 语句之间的水平宽度。与 <IMG> 标记中的 HSpace 的值含义相同
ALIGN	可以是下列值中的任意一个：左、右、顶、文本顶、中间、基准行、底、绝对底	指示小应用程序与页面剩余部分的对齐方式。这些值与 <IMG> 标记中的 ALIGN 的值含义相同
ALT	字符串	如果浏览器不能显示实际的小应用程序，它指定替代的显示文本。该属性只在浏览器能够理解 <APPLET> 标志但不能显示小应用程序时才起作用。否则将显示位于 <APPLET> 和 </APPLET> 标记之间的 HTML 内容
ARCHIVE	文档列表	包含一个文档和在开始执行之前由浏览器预加载的其它资源的列表

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

属性	值	描述
OBJECT	序列化的小应用程序	包含序列化地描述小应用程序的文件的名字。因为系统认为序列化的小应用程序已经被调用，所以小应用程序的 init()就不会被调用，但是 start()方法仍然要被调用。注意：如果有 OBJECT 属性，那么就不需要设置 CODE 属性。两者只要求一个
CODEBASE	统一资源定位器地址	基目录的 URL，小应用程序的类文件都驻留在这里(在安全管理器管理下)。该主机和 HTML 的 < APPLET > 标记指定的主机是小应用程序可以访问的主机

*必要的属性

^强烈推荐的属性

现在看一下程序清单 14.4。以正常体书写的文本是必需的；以斜体书写的是可被替换的，可以被包含在文档中的适当内容所替代。第一行和最后一行是必须的。在这两个标记之间的其它行是可选的。图 14.4 演示了属性是如何影响小应用程序的显示位置的。

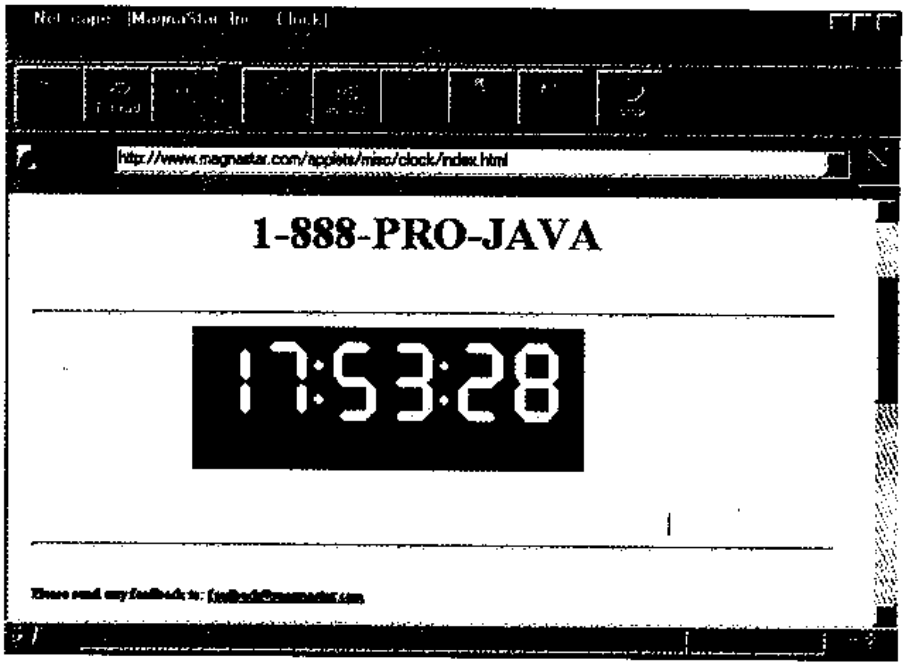


图 14.4 从这个图中可以看出各种属性是如何影响小应用程序的显示位置码

< APPLET *attributes* >

parameters

alternate - content

< /APPLET >

14.4 使用 Java 插件

Sun Microsystem 公司最成功的创新就是增加了众所周知的 Java 插件。Java 插件被设计

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

成为在各种浏览器中具有直接面向 Java 虚拟机的某些特性。在过去的一段时间里各家的浏览器使用了自己的虚拟机。这样一来，各家的浏览器之间总有一些小小的差异。另外 Microsoft 公司没有选择使用 JDK 的某些关键特性。

为了使用 Java 插件，用户必须首先从国际互联网上下载要使用的插件，然后再将插件插入到浏览器中。新加的插件将为浏览器带来对最小的 JDK 的完全支持。通过插入插件，实际完成了对浏览器和虚拟机的升级。这样一来用户就不必担心将来使用的是 JDK 1.2 还是 JDK 的下一个版本了，因为浏览器已经更新了。

14.4.1　在 Internet Explorer 上使用 Java 插件

在 Internet Explorer 上使用 Java 插件需要使用与前面介绍过的 < APPLET > 标记不同的方式。Java 插件实际上与 Internet Explorer 是不同的，需要像包含 ActiveX 部件那样包含它。程序清单 14.5 演示了如何使用在为 Internet Explorer 设计的 HTML 页中使用 Java 插件，并使用了与程序清单 14.3 一样的小应用程序。

```
< OBJECT CLASSID = "clsid:8AD9C840 - 044E - 11D1 - B3E9 - 00805F499D93"
WIDTH = "300" HEIGHT = "300" ALIGN = "aseline"
codebase = "http://java.sun.com/products/plugin/1.1/jinstall - 11 - win32.cab # Version = 1,1,0,0" >
< PARAM NAME = "code" VALUE = "gray.class" >
< PARAM NAME = "codebase" VALUE = "html/" >
< PARAM NAME = "graphic" VALUE = "http://www.magnastar.com/NOW.GIF" >
< PARAM NAME = "link" VALUE = "http://www.magnastar.com/GrayButton/license.html" >
No JDK 1.2 support for APPLET!!
< /OBJECT >
```

程序清单 14.5 中的 < OBJECT > 标记具有许多与程序清单 14.3 中的 < APPLET > 标记相同的属性。同时也有些差别。首先，可能是最明显的一个是 CLASSID 参数。这个 ID 标识的不是 Java 的类，而是 Java 插件。所以用户可能需要找出最新版本号和它的 CLASSID。

用户看到的在 < OBJECT > 标记中的 CODEBASE 不是 Java 类的 CODEBASE，而是 Java 插件的 CODEBAE。因为代码和 CODEBASE 参数都不能在 < OBJECT > 标记中被指定，所以它们将在 < OBJECT > 和 < /OBJECT > 之间的 < PARAM > 标记中结束。

14.4.2　在 Netscape 中使用 Java 插件

与 Internet Explorer 不同，Netscape 没有使用 ActiveX 部件，取而代之的是 Netscape 插件。程序清单 14.6 演示了如何在 Netscape 中使用 Java 插件。

```
< EMBED TYPE = "application/x - java - applet;version = 1.2" WIDTH = "300" HEIGHT = "300"
ALIGN = "aseline" CODE = "gray.class" CODEBASE = "html/"
GRAPHIC = "http://www.magnastar.com/NOW.GIF"
LINK = "http://www.magnastar.com/GrayButton/license.html"
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
pluginspage = "http://java.sun.com/products/plugin/1.1/plugin - install.html" >
< NOEMBED >
No JDK 1.2 support for APPLET!!
</NOEMBED >
</EMBED >
```

使用了 < EMBED > 标记以后,新的参数就不再是 CLASSID 了;取而代之的是 TYPE 值。用户可能已经注意到了类型值是 VERSION。VERSION 允许用户指定用户想使用的 Java 版本。另外要注意 PLUGINSPAGE 参数,它指定当下载不了插件时 Netscape 可以寻找该插件的地方。

关于使用 < EMBED > 标记最有意思的事情是没有与 < PARAM > 标记相应的属性。取而代之的是列在 < EMBED > 标记内部的一些值。注意看一下 LINK 参数,看它是如何使用的。

14.4.3 为所有的浏览器设置 HTML 文本

在程序清单 14.7 中结合使用了 < EMBED > 标记和 < OBJECT > 标记,这样一来 Netscape 和 Internet Explorer 都可以使用 Java 插件了。

```
< OBJECT CLASSID = "lsid:8AD9C840 - 044E - 11D1 - B3E9 - 00805F499D93"
WIDTH = "300" HEIGHT = "300" ALIGN = "aseline"
codebase = "http://java.sun.com/products/plugin/1.1/jinstall - 11 - win32.cab
➡ # Version = 1,1,0,0" > < PARAM NAME = "code" VALUE = "gray.class" >
< PARAM NAME = "codebase" VALUE = "html/" >
< PARAM NAME = "graphic" VALUE = "http://www.magnastar.com/NOW.GIF" >
< PARAM NAME = "link" VALUE = "http://www.magnastar.com/GrayButton/license.html" >
< PARAM NAME = "type" VALUE = "application/x - java - applet;version = 1.1" >
< COMMENT >
< EMBED TYPE = "application/x - java - applet;version = 1.1" WIDTH = "300" HEIGHT = "300"
ALIGN = "baseline" CODE = "gray.class" CODEBASE = "html/"
GRAPHIC = http://www.magnastar.com/NOW.GIF
LINK = ;http://www.magnastar.com/GrayButton/license.html"
pluginspage = "http://java.sun.com/products/plugin/1.1/plugin - install.html" >
< NOEMBED >
</COMMENT >
No JDK 1.2 support for APPLET!!
</NOEMBED > </EMBED >
</OBJECT >
```

但是在程序清单 14.7 中试图让浏览器使用 Java 控制。甚至要求不支持 ActiveX 部件或插件的浏览器也这样做。为了精确起见,最好是使用程序清单 14.8 中的代码。在程序清单

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

14.8 中包含有必要的 JavaScript 代码，该代码驱使浏览器寻找正确的系统。在 Windows 操作系统上的 Internet Explorer 和 Netscape 将使用该 Java 插件，在其它的使用内置虚拟机的平台上也是如此。通常，用户可以把程序清单 14.8 中的代码拷贝到 HTML 文本中并使用正确的参数作一下替换。这样就可以看上去兼容了。用户需要做的事情仅仅是复制和粘贴。

```
<SCRIPT LANGUAGE="JavaScript"><!--
if (_ie == true) document.writeln('<OBJECT
CLASSID="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
WIDTH="300" HEIGHT="300" ALIGN="baseline"
codebase="http://java.sun.com/products/plugin/1.1/jinstall-11-win32.cab#Version=1,1,0,0">
<NOEMBED><XMP>');
else if (_ns == true) document.writeln('<EMBED
TYPE="application/x-java-applet;version=1.1" WIDTH="200" HEIGHT="200"
ALIGN="baseline" CODE="gray.class" CODEBASE="html/"
GRAPHIC=http://www.magnastar.com/NOW.GIF
LINK="http://www.magnastar.com/GrayButton/license.html"
pluginspage="http://java.sun.com/products/plugin/1.1/plugin-install.html">
<NOEMBED><XMP>');
//--></SCRIPT>
<APPLET CODE="gray.class" CODEBASE="html/" ALIGN="baseline"
WIDTH="300" HEIGHT="300"></XMP>
<PARAM NAME="code" VALUE="gray.class">
<PARAM NAME="codebase" VALUE="html/">
<PARAM NAME="graphic" VALUE="http://www.magnastar.com/NOW.GIF">
<PARAM NAME="link" VALUE="http://www.magnastar.com/GrayButton/license.html">
No JDK 1.2 support for APPLET!!
</APPLET></NOEMBED></EMBED></OBJECT>
```

14.5　开始开发 Java 小应用程序

在探究了如何在 HTML 文本中包含小应用程序之后，现在让我们来看一下如何编写自己的小应用程序。

许多年以前，两位程序设计师 Kernie 和 Richie 发明了 C 语言。他们编写的第一个程序叫做 Hello World。从那以后，任何程序员在使用任何语言编写第一个程序时都在屏幕上显示“Hello World”。现在我们也不例外地要使用 Java 编写 HelloWorld 小应用程序。

在前面的几章中，我们已经讨论了 Hello World 应用程序的各部分，现在让我们再来回顾一下它们，如程序清单 14.9 所示。

```
import java.applet.Applet;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.awt.Graphics;
/*
   *
   * HelloWorld
   *
   */
public class HelloWorld extends Applet {
        public void paint (Graphics g){
            g.drawString ("HelloWorld",5,20);
        }
}
```

为了创建 HelloWorld 小应用程序，让我们把程序清单 14.5 中的内容拷贝到一个名为 HelloWorld.java 的文件中。把该文件叫做 HelloWorld 很重要，如果取其它名字就无法编译该程序。假定用户已经把 Sun 公司的 JDK 安装在了命令路径之内了，现在通过在命令提示符下输下列命令来编译该程序：

```
javac HelloWorld.java
```

注意：对于 Windows 用户，为了做这一工作，应该首先打开 DOS 提示符窗口。

如果一切做得都正确，在当前目录中应该增加一个名为 HelloWorld.class 文件。这个文件就相当于传统 DOS 中的 HelloWorld.exe 文件。在运行该文件之前必须创建一个像前面讨论过的 HTML 文件。在这个 HelloWorld 小应用程序例子中，包含它的 HTML 文件应该类似于程序清单 14.10 中的代码。

注意：从技术上讲，类文件本身并不是可执行文件。但是现在有些产品如 Asymetric 公司的 Super Cede 和 Microsoft 公司的 Visual J++ 包含了本地的 Java 编译器，这种本地 Java 编译器可以生成.exe 文件。这种编译器又叫做静态编译器，它能够生成.exe 文件，但是这些.exe 文件就不再是平台无关的程序了。

```
<HTML>
<BODY>
<APPLET CODE="HelloWorld.class" HEIGHT=100 WIDTH=100></APPLET>
</BODY>
</HTML>
```

当创建了 HTML 文件后，用户就可以在像 Netscape Navigator 或 JDK 带的工具之一 appletviewer 这样的浏览器中将它打开。图 14.5 显示了在 Netscape 中加载该文件时出现的情

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

形。

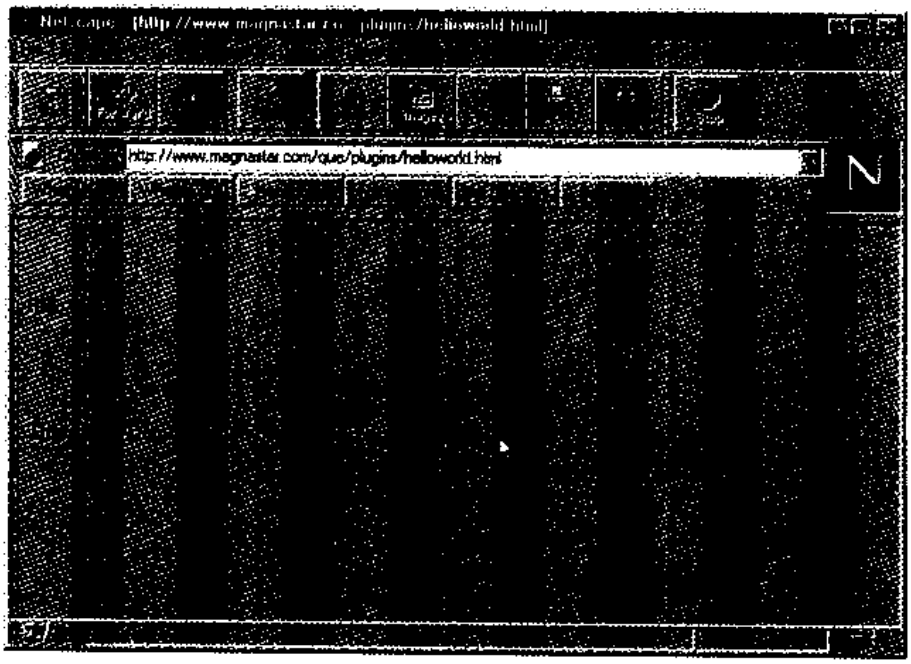


图 14.5　HelloWorld 在浏览器上显示了一些文本

注意，当 Java 小应用程序被加载时，Navigator 在运行 HelloWorld 之前必须回到服务器上（在本例中是回到硬盘上）下载 HelloWorld.class 文件。这和下载 GIF 图像的过程是一样的，但下载 HelloWorld 过程很短，只需 1～2 秒钟。

14.5.1　理解 HelloWorld——构造小应用程序

现在让我们回过头来分解 HelloWorld 小应用程序的代码，这样可以更好地理解它。

第一件需要注意的事情是 HelloWorld 扩展了 java.applet.Applet 类。世界上每一个小应用程序都必须扩展 Applet。正如用户看到的那样，用户正在利用面向对象编程 OOP 的可继承性优点声明自己的小应用程序，使它成为 Java 的 Applet 类的子类。有关继承性的信息请查看第 11 章“类”。

> **注意：**Applet 必须扩展 Applet 的原因是每个浏览器都期望从 CODE 属性中接收到 Applet 类。通过使用继承的多态性特点，用户可以定制小应用程序，比如 HelloWorld，使得它包括 HelloWorld 类和 Applet 类。

14.6　探究小应用程序的生命周期

用户一定会对小应用程序还有生命周期而感到奇怪。小应用程序的生命周期意味着小应用程序存在的时间，即小应用程序的某些方法被调用的时间。更精确地讲小应用程序有四个方法可被调用：

init()　　在加载小应用程序时调用。

start()　　在 init()方法被调用之后被调用，并且在每次浏览器回到包含该小应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的页时都将被调用。

stop() 每当浏览器离开包含该小应用程序的网页时系统调用该方法。

destroy() 当浏览器完全关闭时调用该方法。

图 14.6 显示了小应用程序的生命周期，为了更好地理解小应用程序的生命周期是如何工作的，让我们看一个用于显示方法调用时序的程序。程序清单 14.11 包含了一个每当调用方法时就打印相关信息的程序和由此而建立的方法活动图。

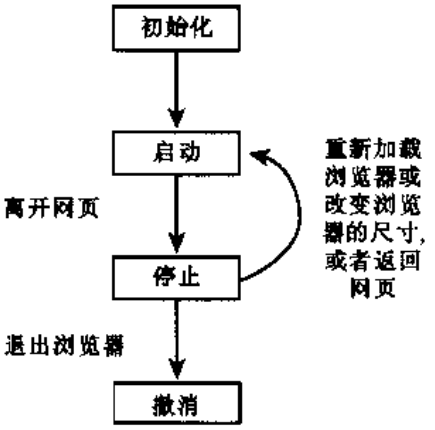


图 14.6 小应用程序生命周期的图形表示

```
import java.applet.Applet;
import java.awt.*;

/*
  *
  * InitStartStop
  *
  */
public class InitStartStop extends Applet{
        int initCount = 0;
        int startCount = 0;
        int stopCount = 0;
        int destroyCount = 0;

        public void paint (Graphics g){
                //clear the area
                g.setColor(Color.white);
                g.fillRect(0,0,size().width,size().height);
                //paint all the standard parts of the graph
                g.setColor (Color.red);
                g.drawLine (120,20,120,220);
                g.drawLine (120,220,300,220);
                //draw the labels
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        g.setColor (Color.gray);
        g.drawString ( "Init Count", 5,50);
        g.drawString ( "Start Count", 5,100);
        g.drawString ( "Stop Count", 5,150);
        g.drawString ( "Destroy Count", 5,200);
        //paint the grid lines
        g.setColor(Color.lightGray);
        for (int x = (120 + 25); x < 300; x + = 25){
            g.drawLine(x,20,x,199);
        }

        //draw the bars for each of the stats
        g.setColor (Color.black);
        g.fillRect (120,30,initCount * 25,40);
        g.fillRect (120,80,startCount * 25,40);
        g.fillRect (120,130,stopCount * 25, 40);
        g.fillRect (120,180,destroyCount * 25, 40);
    }

    public void update(Graphics g){
        paint(g);
    }

    public void init(){
        initCount ++ ;
        System.out.println("init");
        repaint();
    }

    public void start(){
        startCount ++ ;
        System.out.println("start");
        repaint();
    }

    public void stop(){
        stopCount ++ ;
        System.out.println("stop");
        repaint();
    }

    public void destroy(){
        destroyCount ++ ;
        System.out.println("destroy");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        repaint();
    }
}
```

14.6.1 编译 InitStartStop 小应用程序

为了能够运行 InitStartStop 小应用程序，就像 HelloWorld 小应用程序一样，必须对它进行编译并生成一个引用该小应用程序的 HTML 文件。为了实现这些，首先将程序清单 14.11 中的内容拷贝到一个名为 InitStartStop.java 的文件中。然后使用 javac 编译它：

```
javac InitStartStop.java
```

在实际使用 InitStartStop 小应用程序之前，要为它创建一个 HTML 文件。InitStartStop.html 文件如下所示：

```
<HTML>
<BODY>
<APPLET code="InitStartStop.class" HEIGHT=300 WIDTH=400></APPLET>
</BODY>
</HTML>
```

现在可以开始运行 InitStartStop 小应用程序了。先把 InitStartStop.html 文件加载到一个浏览器中，如 Netscape Navigator 浏览器。第一次加载小应用程序后的情形如图 14.7 所示。init()方法和 start()方法都被调用了一次，这是用户能够在图上看到的。

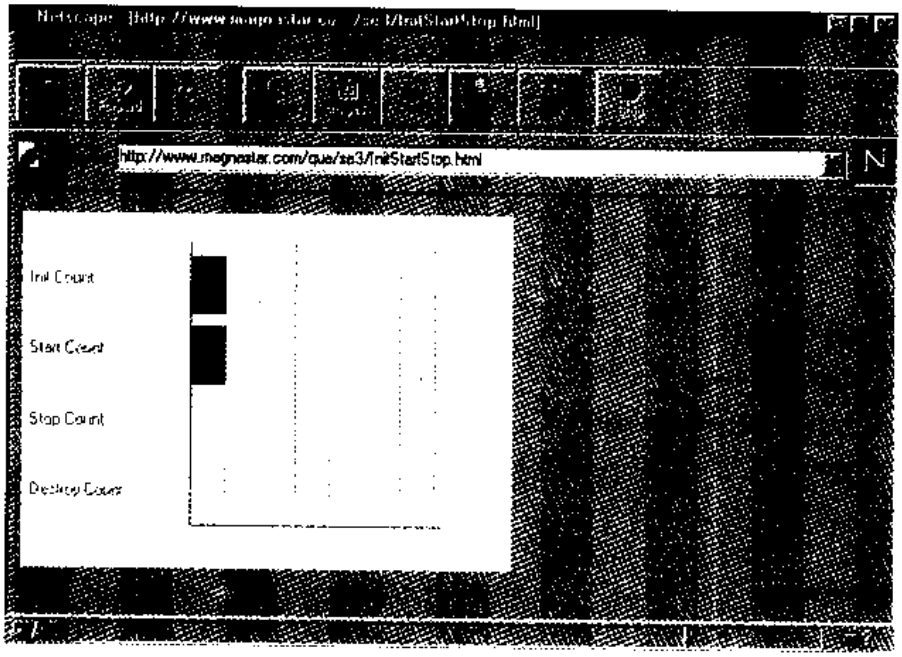


图 14.7 当 InitStartStop 第一次启动时，它先运行一次 init()方法和 start()方法

现在单击两次 reload 按钮。用户每单击一次，stop()方法被调用的次数和 start()方法被调用的次数都被加一(图中增加一个方块)，如图 14.8 所示。init()的记数将停留在 1 的水平上，因为 init()方法只在浏览器第一次加载小应用程序时被调用一次。

在运行小应用程序时，用户也可以看一下打印出来的语句。为了在 Netscape 3.1 或更早

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

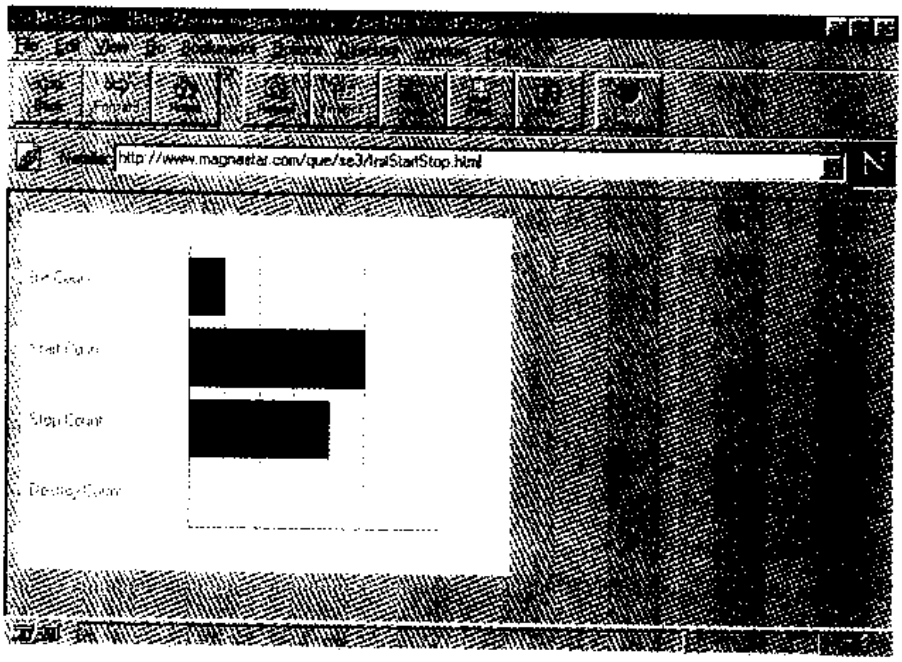


图 14.8　当离开又返回该页几次后，start()方法和 stop()方法被执行的次数都将增长，注意小应用程序的启动次数总是比停止次数多一次

的版本中打印这些语句，可以选择 Option 菜单，显示 Java 控制台。Netscape 4.0 的用户可以通过访问 Communicator(通信员)来打开 Java 控制台。这一操作将打开一个新的窗口，如图 14.9 所示。在这个窗口中用户可以看到所有的 System.out 输出的信息。试着多点几次 reload 按钮。现在再用不同的网页测试一下，看会出现什么结果？当然是 stop()方法被调用，而 Start()方法未被调用。现在如果单击 back 按钮，则会看到 start()方法再一次被调用。

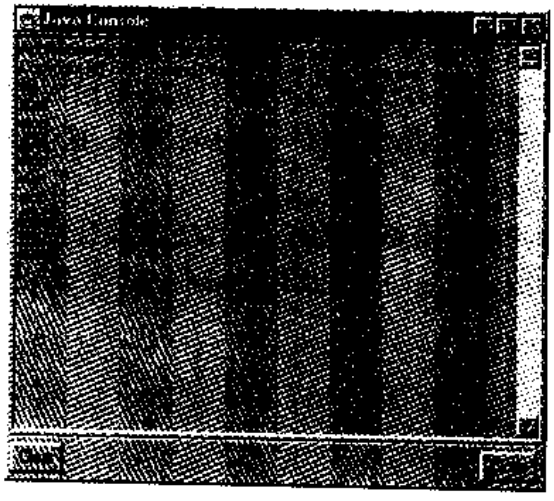


图 14.9　Netscape 的控制台显示了 System.out 的输出信息

14.6.2　理解 InitStartStop 小应用程序

为了理解 InitStartStop 小应用程序，我们一步一步地看它的代码。

```
import java.applet.Applet ;
import java.awt.* ;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

文件的开始就是几行引入语句。正如第 11 章讲过的那样，要使用类(不必每次都全部验证它的名字)，必须首先引入类。就像 HelloWorld 小应用程序一样，InitStartStop 需要访问 java.awt.Graphics 类。另外 InitStartStop 还需要访问其它的两个 java.awt 类。与其单独引入这些类，不如引入整个 java.awt 包。

在 InitStartStop 中的第一个方法是 paint 方法。这个方法使用了 java.Graphics 类中的几个方法在屏幕上画了许多东西。在第 27 章的“文件、流和 Java”中将详细介绍 Graphics 类，那么现在我们就集中注意 Paint()方法的最后部分。

```
//为各个状态画填充矩形
g.setColor(Color.black) ;
g.fillRect (120,30, initCount * 25,40) ;
g.fillRect (120,80, startCount * 25,40) ;
g.fillRect (120,130, stopCount * 25,40) ;
g.fillRect (120,180, destroyCount * 25,40) ;
```

这一部分的目的是画实际的填充矩形，用以显示每种方法被调用的次数。通过增加矩形的宽度来实现计数，宽度为 25 乘以调用次数(比如 25 * initCount)。

```
Public void update (Graphics g){
  Paint(g);
}
```

接下来的方法是 update()方法。Update()方法只是调用 paint()方法，所以用户一定会奇怪这个方法是干什么的。要理解为什么，必须理解 update()和 paint()的关系。一般当小应用程序需要画图时，或者是画在当前显示屏上，或者是显示在小应用程序要显示的位置上都需要调用 paint()方法。然而当小应用程序只需要画出一部分，比如当另一个窗口仅部分地覆盖了小应用程序时，或者当 repaint()方法被调用时，都需要调用 update()方法。缺省情况下 update()方法会清空面板然后调用 paint()方法，但是这会引起屏幕的闪动(试着用这种清除方法运行一下 InitStartStop)。为了避免闪动，应该插入 update 方法，这个 update()方法不做清除，而直接调用 paint()方法。

接下来的几个方法是用户真正需要注意的。每个方法都会使计数器加 1。为了在屏上输入，它们都调用 repaint()方法(repaint()方法调用 update()方法或 paint()方法)。

```
public void init ( ){
    InitCount ++ ;
    System.out.print/n("init") ;
    repaint();
}
public void start(){
    startCount ++ ;
    System.out.println("start") ;
    repaint( );

public void stop ( ){
    stopCount ++ ;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        System.out.println ("stop") ;
        repaint( ) ;
    }
    public void destroy( ){
        destroyCount ++ ;
        System.out.println ("destroy");
        repaint( );
    }
```

14.6.3　Java 动画小应用程序

使用 Java 做的最有乐趣的事情是创建动画。应该指出，Java 不是制作动画的最好工具。如果全部要做的事情都是动画，那么有许多可选用的工具，比如 GIF89a Cel Frame 动画软件。或者可以使用第 44 章“Java 媒体框架”中将介绍的 Java Media Framework。在 Java 中经常要和动画打交道，所以我们在这里演示一个动画程序。程序清单 14.12 显示的是一个完整的 Java 动画程序。

```
import java.awt. * ;
import java.util. Vector;

public class Animator extends java.applet.Applet implements Runnable {
Vector images;
int imgNumber;
int currentImage = 1;
Thread thisThread;

  public void init( ){
    //Read in the number of images in the animation
    imgNumber = new Integer(getParameter("imgNumber")).intValue();

    //Load the images
    for (int x = 0;x < imgNumber;x ++ ){
      Image img = getImage(getDocumentBase(),"images/img" + (x + 1));
      images.addElement(img);
      }
  }

  public void paint(Graphics g){
    g.drawImage((Image)images.elementAt(currentImage ++ ),0,0,null);
    currentImage% = imgNumber;
    }

  public void update(Graphics g){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    paint(g);
  }

  public void start(){
    thisThread = new Thread(this);
    thisThread.start();
  } public void stop(){
    thisThread.stop();
  }

  public void run(){
    while(true){
      try{
        thisThread.sleep(100);
        }
        catch (Exception e){}
    }
  }
}
```

用户可能已经看出这个程序比 HelloWorld 程序复杂了一些。为了编译这个程序，首先将程序清单 14.2 中的全部内容拷贝到一个名叫 Animator.java 的文件中。为了运行它，用户需要为它创建一个 HTML 文件，这个文件如程序清单 14.13 所示。

```
<HTML>
<BODY>
<APPLET code="Animator.class" HEIGHT=200 WIDTH=200>
<PARAM NAME="imgNumber" VALUE="5">
</APPLET>
</BODY>
</HTML>
```

除了这些文件之外，还需要几幅用于动画制作的图像，并且需要将这些图像放置在名叫 image 的子目录下。这些图像必须命名为 image1.gif, image2.gif，等等，其中 imag1.gif 是动画的第一幅画面。用户可能需要将参数 imgNumber 改成正确的图像数。当一切准备就绪后，用户可以得到类似于图 14.10 的显示结果。

为了理解 Animator 是如何工作，现在将程序清单 14.12 分成几个主要的功能块，首先看一下代码的前三行：

```
import java.awt.*;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

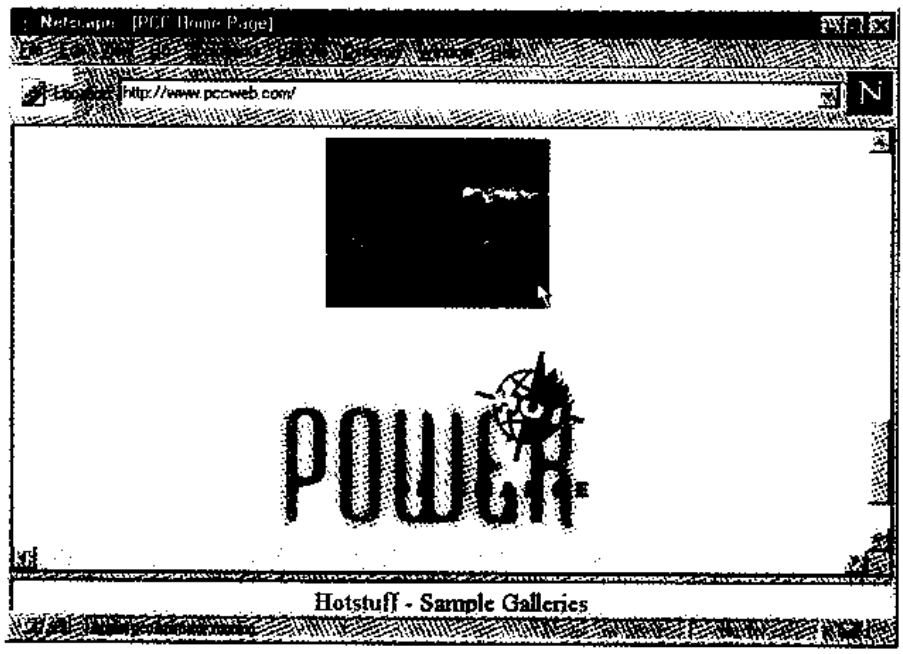


图 14.10　Java 可以用于产生一些有趣的动画

```
import java.util.Vector ;
public class Animator extends java.applet.Applet implements Runnable{
```

头两行用于引入其它 Java 类。Java 是可扩展的语言，该语言的面向对象本性允许用户利用预先构造好的类。头两行的 Animator 代码引入的就是这样的类。

第 3 行代码是类声明，在本行的末尾用户可能会注意到 Animator，像 HelloWorld 一样扩展了 java.applet.Applet 类。java.applet.Applet 是所有小应用程序的祖先，所有小应用程序都是由 java.applet.Applet 扩展的。紧接在类声明后面的是 implements Runnable，它指明了应用程序可以作为线程来运行。Animator 能够作为线程运行非常重要，因为它将在网页被加载完毕后继续运行。

紧接着这三行代码之后，Animator 声明了几个自己拥有的变量。

```
Vector images;
Int imgNumber ;
Int currentImage = 1;
Thread thisThread ;
```

记得在第 10 章“控制流”中讲过：Java 是强类型的语言。这意味着每个变量必须被声明为某个特定的类型。在其它的语言比如 JavaScript 中，用户可以只使用 var 关键字来声明变量：

```
var images ;
var imgNumber ;
var currentImage = 1;
var thisThread ;
```

由于各种原因，这种办法不是最好的做法，Java 要求为每个变量声明类型。正如用户可以看到的那样，在本程序中创建了四个变量。Vector 是一个类型，这种类型便于包含多个元素，特别是当用户不知道要加入多少元素的时候。Thread 变量用于控制小应用程序后来的活动。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Animator 小应用程序有几个方法。其中第一个方法是 init()方法。

```
Public void init ( ){
//读入动画中的图像数
imgNumber = new Integer (getParameter("imgNumber")).intValue( ) ;
//加载图像
for (int x = 0 ;x < imgNumber ; x++ ){
Image img = getImage (getDocumentBase( ), "image/img " + (x + 1);
Image.addElement (img) ;
}
}
```

当页最初被加载到浏览器时调用 init()方法。使用 init()方法设置变量是比较方便的,这些变量只被初始化一次。在 Animator 类中所有的图像只需要调用一次。注意在调用 getImage 方法之后,图像就被加进了图像向量中。

接下来的方法是 paint()方法。Paint()方法在每次小应用程序需要在网页上显示时被调用。当小应用程序被滚出屏幕然后再滚回屏幕时,或者是用户指示重画小应用程序时,就需要调用 paint()方法。

```
public void paint (Graphics g){
g.drawImage ((Image) images.elementAt (currentImage++ ),0,0,null);
currentImage% = imgNumber ;
}
```

我们不分开讲解 paint()方法,只是分析 drawImage 行。drawImage 明显是在图形屏幕上画图像的方法。必须给出 drawImage()方法的四个变量。首先是图像的名字,然后是在 X 方向和 Y 方向上的位置以及 imgObserver,它是针对图像的。

那么为什么图像的名字(Image)images.elementAt (currentImage++)如此复杂呢? 好,现在我们就从右边开始说起。首先用户希望显示当前图像(currentImage)。增加 currentImage 数是很方便的做法,这样就可以使得下次调用时显示的是下一个图像,并且再一次自动地为 currentImage 变量加 1(即 currentImage++)。现在用户已经在向量中存储了这些图像,而从向量中取得当前图像的方法是对图像对象使用 elementAt (即 elementAt(CurrentImage++))。在这一点上唯一的问题是向量并不知道它所包含的图像。向量只知道它包含了某个东西,所以它以不一定正确的方式返回图像,因此用户需要执行所谓的"强制"操作。位于 image.elementAt 前面的(Image)执行的就是强制操作,而这时就可以得到正确的图像。

接下来的方法是 start()方法。每次当用户进入某页时就调用 start()方法。等一等,这时不是调用 init()方法吗? 不,不完全是。用户请看 init()方法,init()方法只在该页被加载时才被调用。这时 start()方法也被第一次调用,但是在第一次使用 init()方法之后,后来的加载只调用 start()方法。

```
public void start ( ){
ThisThread = new Thread(this);
ThisThread.start ( );
}
```

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

start ( )方法会使小应用程序进入一个状态。在 Animator 示例中这个状态就是指线程的建立。这意味着当浏览器的其它部分做其它事情的同时小应用程序可以继续执行。

```
public void stop ( ){
  thisThread.stop( ) ;
}
```

参见第 13 章"线程"。

与 start()方法相对的方法是 stop()方法,每当用户离开该页时 stop()方法就会被调用。当页退出时清除用户启动的东西是非常重要的。Animator 的 stop()方法负责结束正在运行的线程。

Animator 的最后一个方法是 run()方法。run()方法是实际在线程中运行的方法。

```
public void run ( ){
      while (true) {
          repaint( )
        Try {
            ThisThread.sleep (100);
        }
        catch (Exception e){}
    :
}
```

Animator 中的 run()方法执行的是一个循环,它首先告诉 Animator 重画图像,然后休眠 100 毫秒,结果每十分之一秒就用下一幅图像刷新一次动画显示。

14.7　使用控件的小应用程序

正如用户在前面的小应用程序例子中看到的那样,小应用程序是一个交互的应用程序,它可以处理由系统和用户产生的信息。除鼠标以外用户还可以使用在小应用程序中显示的控件,包括按钮、菜单、列表框和文本框与小应用程序交互,如图 14.11 所示。虽然控件将在第 19 章"java.awt:部件"中详细地介绍,但在本章也将简单地介绍它们,现在我们创建一个可以连接国际互联网上各种网站的小应用程序。

程序清单 14.14 是该小应用程序的 Java 源代码,程序清单 14.15 是这个小应用程序的 HTML 文档。在运行这个小应用程序(通过将它的 HTML 文档调入支持 Java 的浏览器)之前应确保计算机已经与国际互联网相联了。然后,当运行该小应用程序时,用户可以看到类似 14.16 所示的显示内容,它显示了在 Netscape Navigator3.1 中运行的 InternetApplet。只要单击某个连接按钮,用户就可以自动地登录到与该按钮关联的网站上。图 14.12 显示了当单击了 CNet 按钮时登录的网站。

```
import java.awt. * ;
import java.awt.event. * ;
import java.applet. * ;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.net.*;
public class InternetApplet extends Applet implements ActionListener {
  boolean badURL;
  public void init() {
    GridLayout layout = new GridLayout(2, 4, 10, 10);
    setLayout(layout);
    Font font = new Font("TimesRoman", Font.PLAIN, 24);
    setFont(font);
    Button button = new Button("Sun");
    button.setActionCommand("http://www.sun.com");
    button.addActionListener(this);
    add(button);
    button = new Button("Netscape");
    button.setActionCommand("http://www.netscape.com");
    button.addActionListener(this);
    add(button);
    button = new Button("Javasoft");
    button.setActionCommand("http://www.javasoft.com");
    button.addActionListener(this);
    add(button);
    button = new Button("Macmillan");
    button.setActionCommand("ttp://www.mcp.com");
    button.addActionListener(this);
    add(button);
    button = new Button("Time");
    button.setActionCommand("http://www.pathfinder.com");
    button.addActionListener(this);
    add(button);
    button = new Button("CNet");
    button.setActionCommand("http://www.cnet.com");
    button.addActionListener(this);
    add(button);
    button = new Button("Borland");
    button.setActionCommand("http://www.borland.com");
    button.addActionListener(this);
    add(button);
    button = new Button("Yahoo");
    button.setActionCommand("http://www.yahoo.com");
    button.addActionListener(this);
    add(button);
    badURL = false;
  }

  public void paint(Graphics g) {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        if (badURL)
            g.drawString("Bad URL!", 60, 130);
    }
    public void actionPerformed(ActionEvent event) {
        String pageName = event.getActionCommand();
        try {
            URL url = new URL(pageName);
            AppletContext context = getAppletContext();
            context.showDocument(url);
        }
        catch (MalformedURLException e) {
            badURL = true;
            repaint();
        }
    }
}
```

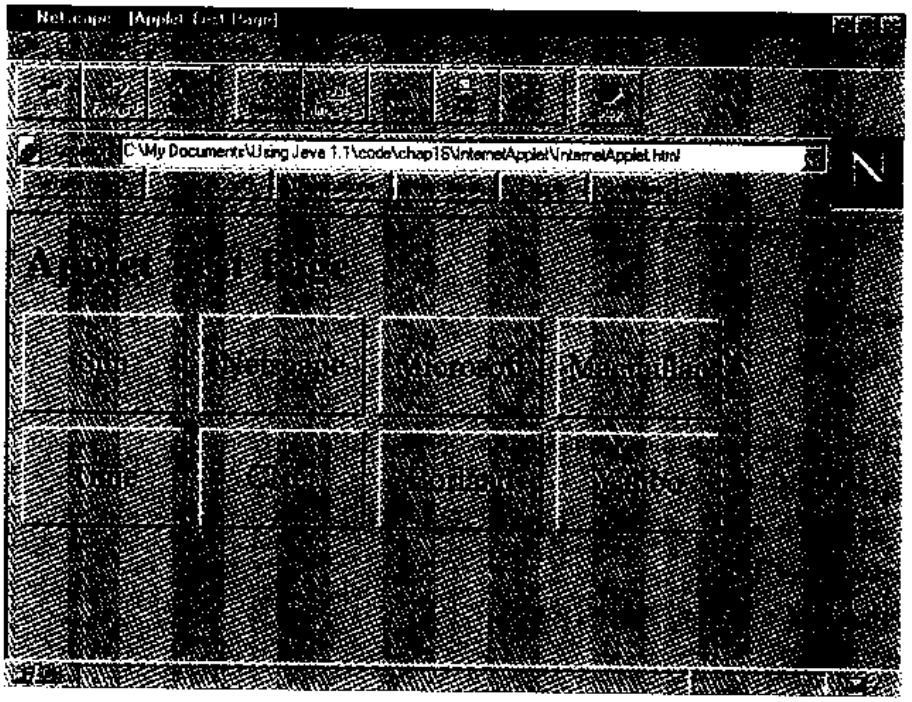


图 14.11　Internet Applet 小应用程序使用按钮提供
与 8 个不同的网页的固定连接

注意：上述的小应用程序只能工作在支持 Java1.1 及以上版本的浏览器中，所以如果用户需要使用尚未升级的老浏览器时，那么就应该看一下程序清单 14.15 中列出的代码。这些代码支持 1.0 模式。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。



图 14.12 当按下 CNet 按钮时,就可以连接到漂亮的 CNet 站点上

```
import java.awt.*;
import java.applet.*;
import java.net.*;
public class InternetApplet extends Applet
{
    boolean badURL;
    public void init()
    {
        GridLayout layout = new GridLayout(2, 4, 10, 10);
        setLayout(layout);
        Font font = new Font("TimesRoman", Font.PLAIN, 24);
        setFont(font);
        Button button = new Button("Sun");
        add(button);
        button = new Button("Netscape");
        add(button);
        button = new Button("Microsoft");
        add(button);
        button = new Button("Macmillan");
        add(button);
        button = new Button("Time");
        add(button);
        button = new Button("CNet");
        add(button);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        button = new Button("Borland");
        add(button);
        button = new Button("Yahoo");
        add(button);
        badURL = false;
    }
    public void paint(Graphics g)
    {
        if (badURL)
            g.drawString("Bad URL!", 60, 130);
    }
    public boolean action(Event evt, Object arg)
    {
        String str;
        if (arg == "Sun")
            str = "http://www.sun.com";
        else if (arg == "Netscape")
            str = "http://www.netscape.com";
        else if (arg == "Microsoft")
            str = "http://www.microsoft.com";
        else if (arg == "Macmillan")
            str = "http://www.mcp.com";
        else if (arg == "Time")
            str = "http://www.pathfinder.com";
        else if (arg == "CNet")
            str = "http://www.cnet.com";
        else if (arg == "Borland")
            str = "http://www.borland.com";
        else
            str = "http://www.yahoo.com";
        try
        {
            URL url = new URL(str);
            AppletContext context = getAppletContext();
            context.showDocument(url);
        }
        catch (MalformedURLException e)
        {
            badURL = true;
            repaint();
        }
        return true;
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 14.16 InternetApplet.html——InternetApplet 的 HTML 文档

```
<TITLE> Applet Test Page</TITLE>
<H1> Applet Test Page</H1>
<APPLET
    CODE = "InternetApplet.class"
    WIDTH = 500
    HEIGHT = 150
    NAME = "InternetApplet">
</APPLET>
```

14.7.1 理解 InternetApplet 小应用程序

现在让我们看一下 InternetApplet 小应用程序的源代码。头三行使得程序能够访问存储在 Java 的 awt 包、applet 包和 net 包中的类：

```
import java.awt.*;
import java.event.*;
import java.applet.*;
import java.net.*;
```

用户已经熟悉了 awt 包和 applet 包。net 包中包括登录国际互联网的类。

小应用程序的主类是从 Applet 类派生出来的类，它以下面的行开始：

```
public class InternetApplet extends Applet
```

然后 InternetApplet 声明它的唯一变量成员：

```
boolean badURL
```

badURL 数据成员在程序中用于通知小应用程序当前选定的 URL 不能用。

14.7.2 探究 init()方法

接下来的是熟悉的 init()方法，在这个方法中小应用程序可以执行一些必要的初始化操作。在本例中，小应用程序首先声明并设置了布局管理器：

```
GridLayout layout = new GridLayout (2,4,10,10);
setLayout (layout);
```

Java 程序使用布局管理器控制程序中部件在屏幕上摆放的位置。Java 提供了许多种布局管理器，每个布局管理器由 awt 包中的类来描述。（关于布局管理器更详细的信息可查看第 21 章“容器和布局管理器”）。如果用户没有创建并设置自己的布局管理器，Java 将使用 FlowLayout 管理器，它缺省地将部件水平地一个挨一个地放置。

在 InternetApplet 中用户使用的是 GridLayout 管理器，它将部件组合在一个格内。GridLayout 的构造器使用四个参数：

- 网格的行数
- 网格的列数
- 网格里单元之间的水平空隙

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

● 单元之间的垂直空隙

如果不为后两个参数赋值，那么它们缺省地使用 0 作为各自的值。

setLayout()方法是 Container 类的成员，Container 类是 Applet 类的超类(在类层次上讲叫做父类)。它的唯一参数是对布局管理器对象引用，当调用完 setLayout ( )方法以后，Java 就知道了要使用新的布局管理器，而不是缺省的布局管理器。

当设置完小应用程序的布局管理器后，程序开始创建和设置将在小应用程序里所有文本中使用的字体：

```
Font font = new Font ("TimesRoman", Font.PLAIN, 24);
setFont (font);
```

Font 类的构造器使用三个参数，字体名、字体属性和字体大小。字体名可以是 Dialog、Helvetica、TimesRoman, Courier 或 Symbol，而字体属性可以是 Font.PLAN、Font.ITALIC 或 Font.BOLD。setFont()方法为小应用程序设置新字体。

接下来的任务是为小应用程序增加用于选择网站的按钮。程序清单 14.17 显示了完成这种任务的代码示例。

```
Button button = new Button("Sun");
button.setActionCommand("http://www.sun.com");
button.addActionListener(this);
add(button);
```

Button 类的构造器只使用一个参数，这个参数就是文本标签，它将在按钮上出现。如果用户希望按钮更丰富多彩，那么用户应该阅读与 JFC 有关的第 34 章“Java 安全性深析”和第 35 章“对象系列化”。JFC 包含许多高级按钮，但是因为它涉及许多其它的内容，所以我们在本章中没有使用。

14.7.3　ActionCommand 和 ActionListeners

当用户创建按钮时，用户应该在选项中指定当按钮被单击时要发出的命令。在缺省情况下，这个命令与放在按钮上的标签相同。但是在本例中，当按钮被单击时用户希望它指向一个 URL(统一资源定位器地址)，所以在每个创建按钮的语句行后面都指定按钮的动作命令。

为了使命令能够使用，用户必须为每个按钮建立一个监听器。当监听器被注册到小应用程序后，它可以接收指示事件发生的通知。在本例中我们希望监听到 Action 事件。

add()方法将按钮增加到 GridLayout 管理器中下一个可用的单元中。

最后，init()将 badURL 标记设置成 false。当输入的 URL 不起作用时。该 bad URL 标记将提示用户。

```
badURL = false;
```

14.7.4　actionPeformed()方法

在用户可以将 this 作为动作监视器加进去之前，用户必须首先实现 ActionListener 接口。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

ActionListener 要求用户创建 actionPeformed()方法。

当用户单击小应用程序的一个按钮时,actionPeformed()方法就会被调用。因为已经为按钮指定 URL 作为命令,所以用户可以使用 action 命令去获取选定的 URL。

```
String pageName = event.getActionCommand( );
```

当获得选定的 URL 后,小应用程序就可以连接网站了。在做这件事情之前,用户必须在程序中设置 try 程序块和 catch 程序块。因为 URL 类的构造器可以产生 MalformedURLException 异常,这个异常必须能够被程序捕获到。(用户在第 20 章"深入研究异常和事件"中可以更多地了解关于异常的知识)try 程序块试图创建 URL 对象并连接到网站上,如程序清单 14.18 所示。

```
try
{
    URL url = new URL(pageName);
    AppletContext context = getAppletContext();
    context.showDocument(url);
}
```

在 try 块中,程序首先从 URL 文本串中创建一个 URL 对象。如果构造失败,URL 类将产生一个 MalformedURLException 异常,并且程序从 catch 程序块开始处继续执行,这个块马上就可以看到。如果 URL 对象创建成功,程序调用 getAppletContext()方法获取对小应用程序的 AppletContext 对象的引用。这个对象的 showDocument()方法将小应用程序连接到选定的 URL 上。

如果 URL 类的构造器产生一个异常,程序将跳到 catch 程序块的开始处继续执行,catch 程序块如程序清单 14.19 所示。

```
catch (MalformedURLException e)
{
    badURL = true;
    repaint();
}
```

在 catch 程序块中,程序简单地把 badURL 设置为 true 并调用 repaint()方法为用户显示一条错误信息。

14.7.5 探究 paint( )方法

程序清单 14.20 显示了小应用程序的 paint()方法,paint()方法不做别的事情,它只是当 badURL 标记被设置为 true 时显示一条错误信息。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:因为 URL 是被硬编码进程序的,所以它不大可能被建错。如果需要修改按钮的 URL,可以打印一条错误信息,好让用户知道输入了不正确的 URL。

```
public void paint(Graphics g)
{
    if (badURL)
        g.drawString("Bad URL!", 60, 130);
}
```

drawString()方法是 Graphics 类的方法,它在屏幕上显示一个文本串。它的三个参数分别显示字符串和字符串的 X,Y 坐标。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 15 章　高级小应用程序代码

本章内容：

15.1　使用 < PARAM > 标记
15.2　在小应用程序中增加图像
15.3　在小应用程序中增加声音
15.4　使用小应用程序控制浏览器
15.5　小结

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

15.1　使用 < PARAM > 标记

Java.applet.Applet 最实用的特性是从 HTML 文件中获取信息。这个信息是非常有用的，因为它允许用户几乎可以把 HTML 文件当做批处理文件来使用，它包含有特定的、小应用程序在运行时刻需要的参数。有了这个信息，用户可以把小应用程序一次写完并定制成人们不熟悉的 Java 代码。这个信息是放置在 < PARAM > 标记中的。

在第 14 章“编写小应用程序”中用户已经知道了 < PARAM > 标记是包含在 HTML 文件中 < APPLET > 标记中的一部分。另外也知道了 < PARAM > 标记的语法是：

< PARAM NAME = "*参数名*" VALUE = *参数值* >

在这里斜体部分可以被具体的信息替代的。在本章中将介绍如何在小应用程序中使用这个信息。

参见第 14 章“编写小应用程序”。

为了访问参数的数据，java.applet.Applet 类提供了一个名叫 getParameter( )的方法。

这个方法的原型是：

```
public String getParameter(String name)
```

正如可以从方法原型中看到的那样，getParameter( )方法只要求一个参数——一个名字。该名字直接对应 < PARAM > 标记中的 NAME。所以如果有如下的标记：

```
< PARAM NAME = "stars" VALUE = 50 >
```

那么，用户就可以得到与下面这行代码类似的结果：

```
String StarCount = getParameter("stars") ;
```

> **注意**：正常情况下，用户实际是先开发程序然后再编写 HTML 文件，所以本例对于用户的实际开发来说可能是逆向的。

如果用户想从参数中得到的值是字符串，那么前面的代码就可以满足用户的需要。但是，本例实际需要的是值为 50 的整数。因为 getParameter( )方法返回的是字符串，那么如何获得整数值 50 呢？答案在于 Wrapper 类中有一个返回 int 的方法叫做 java.lang.Integer。Integer 能够把代表数字的字符串解释成数字值。使用 Integer 类，可以通过 Integer 的 parseInt( )方法把字符串转换成 int 型。如下所示：

```
int starCountInt = Integer.parseInt(starCount);
```

把这几个语句全部放进在屏幕的随机位置上画星星的程序中，就可以得到如程序清单 15.1 所示的代码。

```
import java.applet.Applet;
import java.awt.*;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
/*
 *
 * StarPainter
 *
 */
public class StarPainter extends Applet{
    int starCount;
    public void init(){
        starCount = Integer.parseInt(getParameter("Stars"));
    }

    public void paint(Graphics g){
        g.setColor(Color.black);
        for (int count = 0;count < starCount;count ++){
          int xValue = (int)(getSize().width * Math.random());
          int yValue = (int)(getSize().height * Math.random());
          g.drawLine(xValue,yValue,xValue,yValue);
        }
    }
}
```

在编译完 StarPainter 之后，用户还需要为它创建一个 HTML 文件。在程序清单 15.2 中可以看到 HTML 文件的一个可行的版本。图 15.1 显示了 StarPainter 使用这些参数画出来的结果。试着修改一下星星数，看会出现什么结果(用户可能希望增加一个较大的数值，因为这个小应用程序中小的改变很难看出来)。

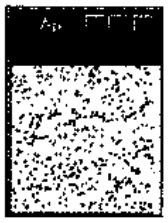


图 15.1 StarPainter 在屏幕的随机位置上画星星

```
<HR>
<APPLET CODE="StarPainter.class" WIDTH=100 HEIGHT=100>
<PARAM NAME=Stars VALUE=500>
</APPLET>
<HR>
```

15.1.1 理解 StarPainter 的源代码

因为用户对小应用程序编程还较生疏，所以还是让我们仔细研究一下程序清单 15.1，完整地看一下 StarPainter 小应用程序是如何工作的。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.applet.Applet ;
import java.awt . * ;
```

第一件需要注意的事情是需要引入几个类，这和所有的 Java 类一样。这实际是很重要的一步。如果不做这种引入就开始写后面的 Java 代码。那么在编译时将遇到下面这样的错误：

```
StarPainter.java:14 Class Graphics not found in type declaration .
  public void paint (Graphics g){
                    ^
```

这个错误就是因为没有引入 Graphics 类造成的。

```
public void init( ){
    starCount = Integer.parseInt(getParameter("stars"));
}
```

StarPainter 的 init()方法应该看上去与我们想象的一样，只是有一点小小的修改。用户已经把在前面看到的两行代码结合到一行上了。注意这说明了把方法(getParameter( ))作为第二个方法(parseInt( ))的参数来使用是完全合法的，这时也可以得到适当的返回值。

```
public void paint (Graphics g){
  g.setColor(Color.black) ;
  for (int count = 0;count < starCount; count ++ ){
    int xValue = (int)(getSize( ).wiclth * Math.random( ) );
    int yValue = (int)(getSize( ).height * Math.random( ) );
    g.draw line(xValue, yValue,xValue,yValue) ;
      }
    }
}
```

现在 StarPainter 的 paint()方法不再包含用户尚未见过的方法了。paint()方法做的第一件事情就是把颜色设为黑色。

参见第 22 章“图形”。

15.1.2　使用 getSize( )方法

现在着重看一下 xValue = 和 yValue = 这两行代码。

```
Int xValue = (int)(getSize( ) .width * Math.random( ) );
```

为了帮助用户理解这一行，现在把它分成对等的几行代码：

```
int width = getSize( ).width;
double randomLoc = Math.random( ) ;
double location = width * randomLoc ;
int xValue = (int) location ;
```

请仔细阅读这个代码段，可以证实它与 StarPainter 中的 Value = 行结果是完全相同的。

代码的第一行使用了用户以前从未见过的方法—getSize( )。getSize( )方法是小应用程序从 java.awt.Component 类中继承来的方法。getSize( )方法在小应用程序中很有用处，因为它

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

允许用户查找出用户可以使用的工作空间。为了理解小应用程序如何获得 getSize()方法，请仔细查看 http://www.javasoft.com/products/jdk1.2/docs/api/packages.html 中的小应用程序 API。

当用户访问该站点时，用户可以看到一个小应用程序的顶层结构，叫做继承树，如下所示：

```
Class java.applet.Applet
 java.lang.Object
   |
   |
   +----java.awt.Component
              |
              |
              +----java.awt.Container
                        |
                        |
                        +----java.awt.Panel
                                  |
                                  |
                                  +----java.applet.Applet
```

继承树可以帮助用户看到所有的 Applet 类是从哪里继承来的。用户可以看到，当通过扩展 java.applet.Applet 类来创建小应用程序时，将获得 Applet 类的所有方法。当小应用程序扩展 Panel 时，小应用程序将获得 Panel 类的所有方法。所以虽然用户没有扩展 Component 类，但是因为 Container 扩展了 Component 类，而 Panel 类扩展了 Container 类，Applet 类扩展了 Panel 类，最后 StarPainter 类扩展了 Applet 类，所以 StarPainter 有效地继承了所有这些类，因此它可以使用这些类中的所有方法。

在继承树的顶部，用户可以看到 Component，它是含有许多方法的类。用户可能希望更详细地了解 Component 类。Component 类拥有一个名为 getSize()的方法，它就是用户正在寻找的方法。

回到前面对等的代码去，width = getSize().width 的下一行是：

```
double randomLoc = Math.random( );
```

Math 是 java.lang 包中的类(即 java.lang.Math)。Math 类有许多很有价值的方法，其中之一就是 random()方法，random()方法返回 0.0—1.0 之间的一个随机数。这是很有用的，因为正像用户在这里使用的那样，它可以产生任意一个用户想要的数。理解了 random()方法和 getSize()方法以后剩下的代码就更容易理解了。

注意：当调用 random()方法时，用户实际上是在 Math 类上调用，不是从实际的 Math 实例上调用。换句话说，用户并没有像下面这样做：

```
Math myMathVar = new Math( );
double randomLoc = myMathVar.random( );
```

为什么用户可以这样做呢？本来用户不能只使用类名调用方法。但是如果方法被定义为静态的，那么就可以不用建立类的实例对象而直接调用该方法。Math 类的所有方法都是静态的，所以用户可以不建立 Math 对象而直接调用 Math 类的方法。如果用户使用 Sun Microsystem 公司的 API 来编辑代码，可以发现 random()方法前面的点是绿色的而不是红色的。这说明 random()是静态的方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

参见第 8 章“方法”。

15.2　在小应用程序中增加图像

建立小应用程序时的另一个常见的任务是显示图像。正如用户在 StaPainter 小应用程序中看到的那样，用户可以使用 Graphics 类来创建自己的图像，另外也可以加载以 .gif 格式和 .jpg格式存储的图像。getImage( )方法就是 java .applet .Applet 类中在小应用程序中加载这些图像的方法。

```
public Image getImage(URL url)
```

getImage( )方法是很容易使用的方法。需要用户做的事情只是指定可以找到该图像的位置的 URL 地址。所以当从 www.magnastar.com 网站上获取名为 banner.gif 的图片时，所有需要用户做的事情是使用类似于下面这行代码的语句：

```
Image testImage = getImage(new URL("http://www.magnastar.com/banner.gif"));
```

把这条语句放在小应用程序中，如程序清单 15.3 所示。

```
import java.applet.Applet;
import java.awt.*;
import java.net.URL;

/*
 *
 * PaintBanner
 *
 */
public class PaintBanner extends Applet{
    Image testImage;
    public void init(){
        testImage = getImage("http://www.magnastar.com/banner.gif");
    }

    public void paint(Graphics g){
        g.drawImage(testImage,0,0,this);
    }
}
```

如果用户想要做的是画一个图像：banner.gif，那么 PaintBanner 就是一个非常有效的小应用程序。然而不可能让这个小应用程序满足用户对 banne.gif 的多种需要。因为 URL 限制了小应用程序，用户可能根本就无法从 www.magnastar.com 网站上加载 banner.gif 图像，除非

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

该小应用程序就驻存在 www.magnastar.com 计算机上。

可以通过使用前面介绍的 getParameter 方法来解决这个问题,可以嵌入一个用作 URL 的参数。因为有了这个参数的限定,所有程序就可以正常工作了。再提一个问题,难道用户希望总是从当前的机器上加载 banner.gif 图像么?换句话说,如果用户创建一个依赖于大量图像的小应用程序,那么当某人在 www.jars.com 计算机上加载这个小应用程序时就需要从 www.jars.com 上获取图像,而不是 www.magnastar.com。

java.applet.Applet 有两个方法可以帮助用户查找这些图像。它们能够告诉用户存放小应用程序有关的类文件的位置的相对 URL 和与包含该小应用程序的 HTML 文件存放位置有关的 URL。这两个方法是 getDocumentBase()和 getCodeBase( )。

```
public URL getDocumentBase( )
public URL getCodeBase( )
```

getDocumentBase()方法返回存放包含小应用程序位置的相对 URL。getCodeBase( )返回存放小应用程序类文件位置的相对 URL。注意关键在于相对的。两个方法返回的都只是文件的相对位置。例如前面介绍过的 banner.gif 的相对 URL 是 http://www.magnastar.com/。如果图像位于名叫 images 的子目录中,那么它的 URL 将是 http://www.magnastar.com/Images。为了得到完整的 URL,用户应当把这两个相对 URL 与实际的文件名联合起来创建一个 URL。如何做到这一点呢?答案有两个;首先用户可以使用 URL 的双参数构造器,比如说 new URL(getDocumentBase( ), "banner.gif")。另外 getImage()自身覆盖也能提供相同的功能。程序清单 15.4 是在程序清单 15.3 基础上使用了 getDocumentBase( )方法。

```
import java.applet.Applet;
import java.awt.*;
import java.net.URL;

/*
  *
  * PaintBanner
  *
  */
public class PaintBanner extends Applet{
        Image testImage;
        public void init(){
            testImage = getImage(getDocumentBase(),"banner.gif");
        }
        public void paint(Graphics g){
            g.drawImage(testImage,0,0,this);
        }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

图 15.2 显示了增加 getDocumentBase( )方法后的结果。

现在，当用户运行 PainteBanner 时，浏览器将在与存放 HTML 文件相同的目录中寻找 banner.gif 图像。当把 PaintBanner 移动到另一个系统上或另一个目录中时，不需要修改 HTML 文件的源代码和 PaintBanner 的源代码。



图 15.2　getDocumentBasc( )方法将返回存放小应用程序位置的相对 URL

注意：getImage()方法的一个有趣的特点是它直接返回。换句话说，程序开始立即处理下一个文件，而不等待 getImage()从网上将图像拖过来。事实上，直到第一次使用该图像之前系统并没有真正把这个图像取回来。用户可能会意识到这个问题，因为图像在屏幕上画得很慢。本书的第 22 章“图形”将详细介绍这方面的知识。

15.3　在小应用程序中增加声音

小应用程序的另一个实用的特性是能够加载和使用声音。Java 2 新增加的特格式，包括 WAVE，AU，AIFF，MIDI 和伟大的 RMF 格式。

声音在 Java 中被抽象成为一个名叫 java.applet.AudioClip 的类。Applet 类提供了几个加载声音文件的方法，getAudioClip()方法可以使用前面提到的各种格式，新的 newAudioClip()方允许用户加载浏览器不支持的声音文件。

```
public AudioClip getAudioClip(URL url)
public AudioClip getAudioClip(URL url ,String name)
public static final AudioClip newAudioClip(URL r)
```

newAudioClip( )方法和 getAudioClip()方法的工作方式与 getImage 方法的工作方式类似。事实上，就像使用 getImage()方法那样，getAudioClip()有两种可能性，一种是 URL，而另一种使用 URL 和名字。为了实际地使用 AudioClip，用户将需要使用 AudioClip 的方法之一：play( )、loop( )或 stop()。这里的每一个方法都是按照用户想象的方式那样工作的。play()方法只播放一次声音片段，loop()方法重复播放声音片段。请慎重地使用 loop()方法。因为人们不希望一遍又一遍地听同一个声音片段，所以不要随意使用 AudioClip 的无止境的循环方法。把这些方法放在一个小的应用程序中，如程序清单 15.5 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.applet.Applet;
import java.applet.AudioClip;
import java.awt.Event;

/*
  *
  * PlayAudio
  *
  */
public class PlayAudio extends java.applet.Applet{
        AudioClip audio;

        public void init(){
            audio = getAudioClip(getDocumentBase(),"welcome.au");
        }

        public boolean mouseDown(Event evt, int x, int y){
            audio.play();
            returns true;
        }
}
```

getAudioClip()方法和 getImagin()方法的主要差别是 getAudioClip()要先到网上然后再返回实际的声音片段。getImage( )方法则不是这样,getImage()直接返回并且只加载用户需要的图像。在用户获取声音片段期间程序的其它部分不管发生什么事情都必须等待声音文件下载完毕方能继续运行。

Applet 类中另一个与声音关系更紧密的方法是 play( )。play()方法是按照与 getAudioClip()方法和 getImage()方法完全一样的方式覆盖的。所不同的是 play()方法获得声音片段后马上就播放。但是它并不存储声音片段,当再一次需要它时,就必须重新从网上下载。

15.4 使用小应用程序控制浏览器

使用小应用程序可以做的另一件事情是提供对浏览器的控制。这是很重要的,因为这种功能允许用户编写的小应用程序能够扩展标准浏览器的功能,这样可以满足用户的特殊需求。

与本章中介绍的其它方法不同的是 java.applet.Applet 类不具有修改网页本身的能力。取而代之的 AudioClip 类是与浏览器连接的,它实际控制着小应用程序的生存环境。

为了使浏览器能够获得 AudioContext,用户需要使用下面的方法:

```
public AppletContext getAppletContext( )
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

当为小应用程序提供了 AppletContext 之后,用户就可以控制浏览器了。

15.4.1　修改状态信息

小应用程序能够引发浏览器修改网页或显示不同的信息。做这种事情的第一个方法是 showStatus( )。showStatus( )能够在状态窗口显示信息,状态窗口一般在页的底部,如图 15.3 所示。

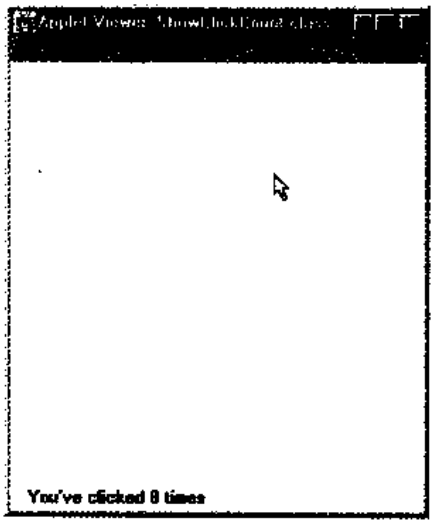


图 15.3　当单击小应用程序时,状态行的内容发生变化

```
public void showStatus(String msg)
```

使用 showStatus(),用户可以将输出的值改成用户想要显示的任何字符串。程序清单 15.6显示了一个示例程序,它显示的状态信息是用户单击小应用程序的次数。

```
import java.applet.Applet;
import java.awt.Event;

/*
   *
   * ShowClickCount
   *
   */
public class ShowClickCount extends Applet
{
        int count = 0;

        public boolean mouseDown(Event evt, int x, int y){
            getAppletContext().showStatus("You've clicked " + (count++) + "times");
            return true;
        }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在运行 ShowClickCount 小应用程序过程中，当用户单击小应用程序时注意状态窗口里状态信息的改变。修改页底部的状态信息是一种很有用的做法。状态信息可以给程序用户一个反馈。注意浏览器是如何使用状态区告诉用户系统将连接到何处或者下载的状态。

15.4.2 修改浏览器显示的页

可以对浏览器做的另一件事情是改变浏览器显示的网页。这也很有用，因为它意味着用户可以为小应用程序增加导航能力。

做这件事情的 AppletContext 的方法有两种形式：

```
showDocument(URL)
showDocument (URL,String)
```

如果用户跳到前面，用户可能会自己想象："哈哈，showDocument()方法与 getImage()方法和 getAudioClip()方法一样有着相同的相对 URL 和最终的文档选项。"如果这是用户正在想象的，很好，然而我没有容易的办法为你分开讲解这个问题，但是我还是要告诉你：你错了。这两个 showDocument()的版本与 getImage()方法和 getAudioClip()方法的工作方式不同；请继续读下去。

首先，showDocument(URL)确实把浏览器窗口修改成用户指定的 URL 的内容，这和用户猜想的一样。所以，如果用户想创建一个修改网页的小应用程序，那么就应该把某些东西像程序清单 15.7 那样放在一起。

```
import java.applet.Applet;
import java.awt.Event;
import java.net.*;

/*
  *
  * ShowDocument
  *
  */
public class ShowDocument extends Applet {
      public boolean mouseDown(Event evt, int x, int y){
          try{
          getAppletContext().showDocument(new URL("http://www.magnastar.com"));
          } catch (MalformedURLException urlException){
              System.out.println("Sorry but there was an error creating the URL:" + urlException);
          }
          return true;
      }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在用户运行 ShowDocument 小应用程序过程中，当单击它时，用户的浏览器将连接到 www.magnastar.com 网页。注意前面示例中 try - catch 程序块。是否理解为什么需要它？这是必须的。因为 URL 的构造器在 URL 无效时产生一个异常。例如，如果用户指向 http://www.magnastar.com，这不可能是一个有效的 URL，因为 URL 不知道用 http 干什么。

那么这两个 showDocument()方法之间有什么不同呢？第一个方法使用一个参数，这个参数在前面介绍过，它是用户想要显示的 URL。第二个方法除了使用用户想要指向的 URL 作为参数外，还使用显示文档将要使用的目标框架的名字作为第二个参数。用户可以用框架的实际名字(如果在网页中使用框架的话)或者是使用值"_self","_parent"和"_blank"分别引用当前框架(缺省值)、父框架或新窗口。

15.5　小结

现在，让我们试着用本章中介绍的所有方法构造一个更复杂的示例。为了使示例完整，首先显示一个图像。然后每当鼠标进入这个小应用程序时，它就播放一个声音。当鼠标单击时显示一个信息，当释放鼠标按钮时，改变页的连接。程序清单 15.8 显示了如何创建这个小应用程序，看用户自己能否完全理解其中的源代码。



```
import java.applet.*;
import java.awt.*;
import java.net.*;

/*
  *
  * ActiveBanner
  *
  */
public class ActiveBanner extends Applet{
      Image banner;
      AudioClip welcome;

      public void init(){
          banner = getImage(getDocumentBase(),"banner.gif");
          welcome = getAudioClip(getDocumentBase(),"welcome.au");
      }

      public void paint(Graphics g){
          g.drawImage(banner,0,0,this);
      }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void update (Graphics g){
        paint(g);
    }

    public boolean mouseEnter(Event e, int x, int y){
        welcome.play();
        return true;
    }

    public boolean mouseDown(Event e, int x, int y){
        getAppletContext().showStatus("Release the mouse button to go to MagnaStar");
        return true;
    }

    public boolean mouseUp(Event e, int x, int y){
        try{
            getAppletContext().showDocument(new URL("Http://www.magnastar.com"));
        } catch (MalformedURLException urlException){
            System.out.println("Sorry but there was an error creating the URL:" + urlException);
        }
        return true;
    }

}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 16 章　JAR 档案文件

本章内容：

16.1　为什么使用 JAR
16.2　什么时候使用 JAR 档案
16.3　JAR 档案和安全性
16.4　java.util.zip 包
16.5　JAR 文件格式

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

16.1 为什么使用 JAR

JAR 文件格式为小应用程序带来几个优点。这些优点包括执行效率的提高和可移植性的提高。JAR 文件也实现了从 JDK 1.1 开始引入的 Security Model。Security Model(安全性模式)将在第 34 章"Java 安全性深析"中详细介绍。

JAR 档案并不是 Java 支持的第一个档案格式。从版本 1.0 开始,JDK 就使用未压缩的 ZIP 文件 classes.zip 把 JDK 的系统类文件存储成单一的磁盘文件。Netscape Navigator 3.0 遵照这个做法,允许 <APPLET> 标记从类似的 ZIP 文件中加载小应用程序。从 3.0 版本开始,Microsoft 公司的 Internet Explorer 开始允许从 Microsoft 公司的 CAB 格式的文件中像加载 ActiveX 控件一样加载 Java 小应用程序。

JAR 文件正在取代其它的这些机制。JAR 文件提供的下述好处使它成为广大用户优先的选择。

16.1.1 绑定

一个复杂的小应用程序可能由数十个或数百个 Java 类组成,每个类都分别地存放在各自的类文件中(可以回忆一下本书前面介绍过的内容,每个公共的类必须存放在单独的文件中)。为了运行小应用程序,当需要通过服务器加载文件网页时,浏览器必需通过建立与 HTTP 的连接来加载每个文件。必须首先建立与 HTTP 的连接,如果类文件很小(一般都很小)多次加载小应用程序就需要建立多个 HTTP 连接才能加载所有的类文件。

JAR 文件第一个最明显的好处是把几个类文件结合到一个档案文件中,它可以通过一个 HTTP 连接把所有的类文件都从服务器传输到网页浏览器上。另外,JAR 文件不仅仅能够包含类文件,而且可以包含声音文件和图像文件,它允许将整个小应用程序通过一次交互就全部下载完毕。

16.1.2 压缩

JAR 文件与 CAB 文件一样(但是与 classes.zip 不同),是使用标准的 Lempel - Ziv 算法的变体压缩的文件。例如 JDK 的 TicTacToe 演示的 JAR 文件比原来减少了 20%;ImageMap 演示的 JAR 文件比原来减少了 5%(ImageMap 含有很多图像文件,这些图像文件已经被压缩过)。因为 JAR 不仅仅汇集了许多文件,而且还把这些文件做了压缩处理,所以 JAR 文件可以大大地减少下载小应用程序的时间。

16.1.3 向下兼容性

因为 JAR 档案保留了文件的目录层次,并且可以通过对 <APPLET> 标记做简单的修改就可以对它们进行调用,所以 JAR 文件可以被已有的小应用程序透明地加以调用,不需要改变原来的小应用程序的代码。

16.1.4 可移植性

在这种情况下,可移植性包含两种含义:在浏览器之间的可移植性和在网络服务器之间的可移植性。Netscape Navigator, Microsoft Internet Explorer 以及其它浏览器之间的不兼容性

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

对于任何开发网页和 Java 小应用程序的人都是一件头疼的事情。在 Java1.1 之前网络开发者没有可移植的档案机制。

JAR 文件通过提供一个单独的与浏览器无关的档案文件格式解决了这个问题。因为 JAR 支持 Java,并且工具全部是使用 Java 实现的,所以任何支持标准的 Java1.1 + 库的浏览器都能够支持 JAR 文件。

可移植性问题的另一方面即当用户试图将小应用程序从某个网络服务器移动另一个网络服务器上,这个问题就变得很清楚了。例如,想象用户开发了一个运行在基于 Windows 95 的网络服务器上,用户的文件使用了具有描述意义的名字,比如 NavigationBarAnimationPanel.class,这个名字在 Windows 95 下是合法的。现在用户需要将该网站移到使用 Macintosh 操作系统的网络服务器上。不幸的是用户将会发现 Macintosh 的文件名是被限制在 31 个字符之内的,用户不仅需要把源文件重新命名,而且需要把文件中的类(因为文件名必须与它们包含的类的名字匹配)也重新命名。

(为了看到第一手材料,应该试着下载 JDK 1.1 的 beta2 版的文档文件并安装在 Macintosh 系统上,许多文件名会被截断,并且浏览器无法引导链接到这些文件上。)

通过将小应用程序的各种类文件和其它的资源文件存储在单一的 JAR 文件中,可以消除由各种网络服务器低层文件系统的差异产生的影响。

16.1.5　安全性

对于 JDK 1.1 来说,Java 的安全模式已经被扩展了。现在通过使用授权了的 JAR 文档,可以为用户验证小应用程序的来源,从而将它标记为可信任的并且给予它更多的特权。这就使编写新型的小应用程序成为可能,例如在本地用户的硬盘上存储文件的字处理器。

16.2　什么时候使用 JAR 档案

如果用户的小应用程序遇到下列情况,就应该考虑使用 JAR 档案:

- 用户希望减少小应用程序的加载时间,特别是在小应用程序是由许多文件组成的时候。
- 用户希望简化自己的小应用程序时,或者是希望使小应用程序可移植到更多的网络服务器时。
- 用户的小应用程序需要被作为信任码授权时。

提示:JAR 文件主要是对小应用程序有用。如果用户正在开发 Java 应用程序,JAR 文件就没有什么用处了,尽管用户可能仍然将它们作为通用的文件格式来使用,但是没有什么特殊的含义。

16.2.1　jar 工具

jar 工具允许用户从 JAR 档案中创建、列表和扩展文件。从功能和用法上它非常像 UNIX 中的 tar 工具。与 JDK 中的其它工具一样,jar 工具是以应用程序的形式实现的,这使它可以移植到任何支持 Java 的平台上。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

创建新档案 为了创建新档案,需要使用 cvf 选项。其中 c 选项告诉 jar 创建一个新的档案。v 选项告诉 jar 输出冗余的诊断信息给控制台,这样可以使用户能够看到正在加入的内容。f 选项告诉 jar 使用给定的名字创建档案文件。例如:

```
jar cvf Foo.jar * .class images
```

这一行命令将在当前目录中创建名为 Foo.jar 的新 JAR 档案。该档案将包括当前目录中的所有类文件和 image 目录以及该目录中的全部内容。

作为一个例子,我们连接到包含 JDK 演示示例 TicTacToe 的目录上。

目录里内容的列表显示了一个类文件和包含声音文件与图像文件的两个子目录:

```
D: \ java \ demo \ TicTacToe > dir
Volume in drive D is NTFS20
Volume Serial Number is 6C98 - 56B4
Directory of D: \ JDK 1.2 \ demo \ TicTacToe
01/13/97  10:04a      < DIR >                .
01/13/97  10:04a      < DIR >                ..
12/16/96  11:29a      < DIR >                audio
11/19/96  12:34p                   139 example1.html
12/16/96  11:29a      < DIR >      images
11/19/96  12:34p                 3,454 TicTacToe.class
12/06/96  10:27a                 7,593 TicTacToe.java
              7 File(s)             11,186 bytes
                        1,575, 772,160 bytes free
```

创建一个包含该小应用程序的 JAR 文件版本的新子目录:

```
D: \ JDK 1.2 \ demo \ TicTacToe > mkdir jar
```

现在创建 JAR 档案:

```
D: \ JDK 1.2 \ demo \ TicTacToe > jar cvf  jar \  TicTacToe.jar * .class audio images
adding: TicTacToe.class
adding: audio/beep.au
adding: audio/ding.au
adding: audio/return.au
adding: audio/yahool.au
adding: audio/yahoo2.au
adding: images/cross.gif
adding: images/not.gif
```

注意当目录以输入文件的形式列给 jar 工具时,目录中的内容就被加进了档案,并且保留了目录名。

当 jar 工具创建新的档案时,它自动地增加一个声明文件到文档中,在大多数情况下,这样做就足够了。但是有时用户希望创建自己的声明文件,在这种情况下用户必须在使用 jar 工具的时候指定的 m 选项。

列出文档内容 jar 工具也可以列出 JAR 档案的内容。例如:

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
jar tvf Foo.jar
```

这个命令将列出 Foo.jar 的内容。

为了继续介绍 TicTacToe 演示小应用程序，我们连接到用户前面创建的 jar 子目录上。使用 t 选项可以获得一个列表。

```
D:\JDK 1.2\demo\ TicTacToe\jar tf TicTacToe.jar
META-INF/MANIFEST.MF
TicTacToe.class
audio/beep.au
audio/ding.au
audio/return.au
audio/yahoo1.au
audio/yahoo2.au
images/cross.gif
images/not.gif
```

注意声明文件已经被自动地加进了档案。有关声明文件更详细的信息，用户可查看本章后面的“声明文件”部分。使用 v 选项用户可以获得更多的信息：

```
D:\JDK 1.2\demo\ TicTacToe\jar>jar tvf TicTacToe.jar
1045 Mon Jan 13 11:52:18 PST 1997 META-INF/MANIFEST.MF
3454 Tue Nov 19 12:34:26 PST 1996 TicTacToe.class
4032 Tue Nov 19 12:34:26 PST1996 audio/beep.au
2566 Tue Nov 19 12:34:26 PST1996 audio/ding.au
6558 Tue Nov 19 12:34:26 PST1996 audio/return.au
7834 Tue Nov 19 12:34:26 PST1996 audio/yahoo1.au
7463 Tue Nov 19 12:34:26 PST1996 audio/yahoo2.au
157 Tue Nov 19 12:34:26 PST1996 images/cross.gif
158 Tue Nov 19 12:34:26 PST1996 images/not.gif
```

从档案中抽取文件　最后，jar 工具可以从档案文件中抽取文件。例如，为了抽取 TicTacToe.class 文件，可以使用下面的命令：

```
D:\DJK1.2\demo\ TicTacToe\jar>jar xvf TicTacToe.jar TicTacToe.class
extracted: TicTacToe.class, 3454 bytes
```

如果用户紧接着要在用户的计算机上删除刚刚抽取的文件以便接下来的例子能够正常工作，那么用户可以使用下面的命令：

```
D:\JDK 1.2\demo\ TicTacToe\jar>del TicTacToe.class
```

提示：用户不能使用 x 选项从 JAR 档案的子目录中抽取单个文件。正确的方法是抽取完整的子目录并在取出后删除不需要的文件。

16.2.2　<APPLET>标志

<APPLET>标志可以将 Java 小应用程序嵌入到 HTML 文件中。小应用程序拥有许多

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

属性，这些属性可以指定要加载的小应用程序的名字，用于定位要加载的小应用程序的 URL 和小应用程序在页面上占用的位置等。除了这些属性以外还有许多参数可以在这里指定。例如下面的 HTML 文件：

```
< APPLET CODE = "FooMain.class" WIDTH = 100 HEIGHT = 120 >
< PARAM NAME = "color" VALUE = "red" >
< PARAM NAME = "background" VALUE = "blue" >
< /APPLET >
```

CODEBASE 属性指定加载文件来源的 URL。如果 URL 未被指定，则使用引用页的 URL。例如，在运行下面这个 HTML 文件时，浏览器将试图从 http://www.foo.com/applets/FooMain.class 处加载紧接其后的小应用程序：

```
< APPLET CODE = "FooMain.class" CODEBASE = "http://www.foo.com/applets/" WIDTH = 100 HEIGHT = 120 >
...
< /APPLET >
```

从 JDK 1.1 开始，Sun 公司把这个能从 JAR 档案中加载类的 < APPLET > 标志修改为在定位 Java 小应用程序类之前下载 JAR 档案中的类。

从 JAR 档案中加载可以使用两种指定方式：使用属性或使用参数。第一，可以使用名为 ARCHIVES 的属性，例如：

```
< APPLET ARCHIVES = "Foo.jar" CODE = "FooMain.class" >
...
< /APPLET >
```

当浏览器读到这个标记时，它首先从服务器下载 Foo.jar 文件，然后试图在 Foo.jar 中寻找 FooMain.class。如果浏览器在档案中找不到该类，那么通常它将在由 CODEBASE 指定的位置上寻找。

另外，JAR 档案可以被指定为某个参数。这个参数应该含有档案名 ARCHIVES。该参数的名字就是 JAR 文件的名字，例如：

```
< APPLET CODE = "FooMain.class" >
< PARAM NAME = ARCHIVES VALUE = "Foo.jar" >
...
< /APPLET >
```

在小应用程序标记之间指定 JAR 档案是比较有效的优化方案，可以指示浏览器预加载某个指定的档案和使用该档案，如果可能的话，在定位类的同时就执行上述操作。如果 JAR 文件没有找到。或者档案中需要类没有找到，那么就将执行由 JDK1.0 定义的常规搜索过程。指定 JAR 文件执行预加载并不会防碍必要的常规搜索的执行。

作为最后一个例子，让我们查看一下由 JDK 1.1 里的 TicTacToe 演示使用的 < APPLET > 标记。examplel.html 文件中包含有这个小应用程序标记，如程序清单 16.1 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
< title > TicTacToe < /title >
< hr >
< applet code = TicTacToe.class width = 120 height = 120 >
< /applet >
< hr >
< a href = "TicTacToe.java" > The source. < /a >
```

将该文件拷贝到前面创建的 jar 子目录中。

```
D: \ JDK 1.2 \ demo \  TicTacToe > copy example1.html jar
        1 file(s) copied
```

现在编辑该文件并加入 APPLETS 属性。当做完之后程序应该如程序清单 16.2 所示。

```
< title > TicTacToe < /title >
< hr >
< applet code = TicTacToe.class archives = TicTacToe.jar width = 120 height = 120 >
< /applet >
< hr >
< a href = "TicTacToe.java" > The source. < /a >
```

现在用户可以从前面创建的 JAR 档案中运行 TicTacToe 小应用程序了。

```
D: \ Java \ demo \  TicTacToe \ jar > appletviewer example1.html
loading d: \ jdk11b2 \ java \ bin \ . . . lib \ awt.properties
```

16.2.3　兼容的浏览器

为了使用 JAR 文件，用户必须使用 JDK 1.1 浏览器，这意味着如果用户加入了 Java Activator，那么用户就可以使用 Navigator 3.0 或 Internet Explorer 3.0。如果使用的是 4.0 或 4.0 以上版本的浏览器，那么就不必担心了。因为这些版本中已经包含了 Java Activator。

16.3　JAR 档案和安全性

网页允许内容被下载，而 Java 的结构扩展了网页，它允许可执行内容被下载。虽然这一点开拓了广阔的可能性，同时它也开放了许多新的风险。静态的文本和图像文件只可能对接受者造成几乎为零的危害（就像雪花打在人身上一样），但是一段代码极有可能造成巨大的伤害，计算机病毒就是一个很好的见证。

为了保护下载代码的接受者，Java 使用了名为 sandbox（沙箱）的安全模式。sandbox 是一个范围，不被浏览器信任的 Java 代码可以在这个范围内做它想做的事情。通过把小应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的活动限制在定义好了的区域内,浏览器可以在保护 sandbox 之外的一切的同时运行不信任的小应用程序。受保护的内容包括本地机器的内存、文件、磁盘和网络。

在 sandbox 之内运行不信任代码并不会阻碍小应用程序显示时钟、堆栈刻度和动画导航条。但是当小应用程序实现字处理功能或电子表格功能时该怎样保护呢? 这种小应用程序是很有用的,但是它为了读取文件和写入文件(除非小应用程序希望在远程服务器上维护用户数据,这种处理方法很复杂),就需要与用户的本地机器打交道。为了做到这一点,小应用程序需要离开沙箱(sandbox)。在 JDK 1.0 中,小应用程序很难实现这种做法。但是通过使用 JAR 文档,小应用程序有一个标准的方法很容易获得信任状态。

16.3.1 声明文件

JAR 文件的第一项就是一个关于该文档的元信息集成。jar 工具自动地产生元信息并把它存储在顶层目录中名为 META-INF 的文件里。这个目录总是包含声明文件 META - INF/MANIFEST.INF(如清单 16.3 所示)。

```
Manifest - Version: 1.0

Name: TicTacToe.class
Hash - Algorithms: MD5 SHA
MD5 - Hash: TsjcL1vWU7k4/HDkwOnvHg = =
SHA - Hash: IGRKfYKD8Cpef7 + or5ZKqYp3bh0 =

Name: audio \ beep.au
Hash - Algorithms: MD5 SHA
MD5 - Hash: kZv279ZIA/H6mOw4t8W8XA = =
SHA - Hash: JgfdUl4/uzNq5yUy3e07ZXwvNOc =

Name: audio \ ding.au
Hash - Algorithms: MD5 SHA
MD5 - Hash: 23oJDEp/LqCZC70AEIOsVQ = =
SHA - Hash: dpRUB8DKzEP0Grc7DIrXclPMjJ8 =

Name: audio \ return.au
Hash - Algorithms: MD5 SHA
MD5 - Hash: tBUwkF2qeyor/nmPeF81hg = =
SHA - Hash: ABV7ArlgRYQmpp7kSbkH3GN + YOA =

Name: audio \ yahool.au
Hash - Algorithms: MD5 SHA
MD5 - Hash: Bq9PhKz6zAWrgQvtCWS8zQ = =
SHA - Hash: qU03jWxRvJWIp25S9XRQk5lbLaY =
```

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
Name: audio \ yahoo2.au
Hash - Algorithms: MD5 SHA
MD5 - Hash: 6lhsclKkFy5iBu + km + DAVQ = =
SHA - Hash: Gfc7hOmtTmM31JJlHJZgkMm2elo =

Name: images \ cross.gif
Hash - Algorithms: MD5 SHA
MD5 - Hash: gTJaDGQtdz1Y4W + hHWxjgA = =
SHA - Hash: plA3I8zo53u8XXj9 + vutZupQo0U =

Name: images \ not.gif
Hash - Algorithms: MD5 SHA
MD5 - Hash: SJsp04DooHqq9ndFnn6S6w = =
SHA - Hash: MmqEk9R8pMigNK3xDi2yK1cyyZ8 =
```

在正常情况下，如果不使用授权，那么声明文件中将包含档案中其它文件的检验和。例如，用户可以从前面创建的 TicTacToe 档案中抽取声明文件，如下所示：

```
D: \ JDK 1.2 \ demo \ TicTacToe \ jar > jar xvf TicTacToe.jar META-INF extracted: META-INF/MANIFEST.MF, 1045 bytes
```

声明文件列出了档案中的所有文件，以及标签值 MD5-Hash 和 SHA-Hash。清单 16.3 显示了一个典型的声明文件。MD5 和 SHA 是信息摘要，也叫做单向散列函数。散列函数接收一个任意的输入数据片段作为参数，并产生一个固定长度的输出数据片段。MD5 散列是 128 位的；SHA 的散列值是 160 位的。单向的意思是从两个不同的输入很难产生同一个散列值。

manifest 中的信息摘要可用于确认该档案没有被意外地损坏。当浏览器从档案中读取每一个文件时，可以计算出文件的 MD5 和 SHA 散列值并把它们与文件中的值比较以此验证档案有没有被破坏。但是从另一方面讲，这种验证并不能排除蓄意的破坏，因为当某人蓄意地破坏档案文件时，他也可能会同时修改声明文件中的相应散列值。但是发现 JAR 档案中的文件被蓄意地破坏是完全有可能的。为了侦查这种破坏。JAR 档案必须有“签名”。这和用笔在纸上的文件签名是类似的。它确定地表明给定的文件来自于确定的来源。事实上数字签字强于物理的签名，它很难伪造，它不能被签字者复制，并且签过字的文档不能被修改。

16.3.2　私有密钥、公共密钥和证书

为了签署 JAR 档案，用户必须首先创建私有密钥、公共密钥和证书。私有密钥和公共密钥是一对数据片段，用来创建数字签名和加密数据。证书是由某个实体提供的保证，这个实体通常是可信任的公共组织，它保证另一个实体的公共密钥是有效的（在本论题中更具体地说就是证书要证实符合由 CCITT 发布的 X.509 标准）。公共密钥和证书的结合可以用来验证数字签名。

16.3.3　密钥工具

Keytool 工具处理身份、公共密钥、私有密钥和证书的创建和管理。密钥和证书的创建和管理的细节超出了本章的介绍范围，本书将在第 34 章中详细介绍它们。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Keytool 程序可以很快地创建文件名为 Keystore 的数据库。这些数据库实际就是一些文件,这些文件驻存在 JDK 安装的根上,它们包含有用户创建或使用的证书。

Keytool 本身有一些参数,用于指定密钥的操作。现在我们看到的就是一个方案,即产生密钥。为了产生密钥,用户必须知道几件事情。首先,为了记住它用户需要为它起一个别名。现在我们使用 javajoe 作为别名。然后用户需要识别别名以表明用户的身份。这个名字是 X.509标准的一部分,用于指明用户的名字,下面是格式:

```
CN = commonName OU = organizationUnit O = organizationName L = localityName S = stateName C = country
```

这里的每个域都有助于确认用户的身份;例如,我的 - dname 可以是:"CN = Joe Weber, OU = QUE, O = Macmillan Publishing, L = Milwaukee, S = Wisconsin , C = US"

现在用户可以使用这些值生成一个密钥:

```
keytool -genkey -dname "CN = Joe Weber, OU = QUE, O = Macmillan Publishing, L = Milwaukee, S = Wisconsin, C = US" -alias javajoe
```

正像用户可能猜出来的那样, - genkey 命令告诉 keytool 用户正在生成一个新的密钥。-dname指定识别名,-alias 指定用户将要使用的别名。注意别名是大小写敏感的,所以 javajoe 与别名 JavaJoe 是不相同的。

当用户像这样运行 keytool 时,要求用户分别为 keystore 和新密钥输入口令。以后每当用户使用密钥时系统都要提问口令。用户可以在命令行中使用 - storepass - keypass 参数指定密钥口令。如果用户愿意,可以使用下列命令把 - storepass 设置为 mystorepassword,把 - keypass 设置为 privatekeypassword:

```
keytool -genkey -dname "CN = Joe Weber, OU = QUE, O = Macmillan Publishing, L = Milwaukee, S = Wisconsin, C = US" -alias javajoe - storepass mystorepassword - keypass privatepassword
```

通常情况下生成密钥使用的参数是:

```
keytool -genkey [-alias alias] [-keyalg keyalg] [-keysize keysize] [-sigalg sigalg] [-dname dname] [-keypass keypass] [-validity valDays] [-keystore keystore] [-storepass storepass] [-v]
```

16.3.4 jarsigner

既然用户已经生成了一个密钥,那么用户就可以对 JAR 文件进行数字化签名了。签署文件是非常有用的,这样用户和文件的使用者就可以确保用户是文件的发送者并且保证文件不被篡改。

用户在签署 JAR 文件之前,需要知道两个细节。首先要知道用户希望使用的别名。其次要知道 - keystore 口令和用户将要使用的密钥和私有密钥口令。最后,用户可能需要知道 keystore 文件的位置。如果用户把它放在了缺省的位置,就不需要管这事情,但是如果用户把它移动到了别的位置,就需要指明这一信息。

通过使用用户在"密钥工具"部分创建的密钥,现在可以使用下面的命令行来签署 TicTacToe.jar 文件了:

```
jarsigner - storepass mystorepassword - keypass privatekeypassword TicTacToe.jar javajoe
```

当用户签署 JAR 文件时,在声明中增加了两文件。第一个文件是一个 .SF 文件。.SF 文件中包含的信息非常类似于声明文件,它总是被包含在 JAR 文件中。但是 .SF 文件的摘要不包

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

含文件中的二进制数据的散列(和声明一样),但是包含声明中的数据散列。它锁定在声明信息中。

第二个文件是.DSA 文件。.DSA 文件包含.SF 文件的签名并且也包含一个压缩的.SF 文件和与用于签名的私有密钥相应的公共密钥的授权证书。

哇,这词句也太长了。幸运的是,用户没有必要知道这其中的细节。但是用户应该知道这两个文件别名的前 8 个字符(转换成大写字符);在这里它们分别是 JAVAJOE.DSA 和 JAVAJOE.SF。用户可以多次签署 JAR 文件,然后把签名连起来,这样就可以为每个签文件的人生成一个.SF 和一个.DSA。

jarsigner 有许多附加的选项,根据不同的情况用户可以使用其中的选项。现在这些选项概括如表 16.1。

表 16.1　jarsigner 选项

选　项	描　述
-keystore 文件	指定 kestore(数据库文件)的位置。缺省情况下,该文件指的是在用户的主目录中的.keystore文件。这个目录由系统的 user.home 属性指定。在 Windows 系统中,user.hom是由 HOMEDRIVE 和 HOMEPATH 环境变量连接后指定的路径,如果它们连接后生成不了有效的路径,那么将指向 JDK 安装目录的根
-storepass 口令	指定 keystore 口令。用户只在签署 JAR 文件时需要这个口令,在验证 JAR 文件时不用。如果用户没有指定该命令选项,系统将提示用户输入它。出于安全的缘故,一般在命令行中都不指定这个口令
-keypass 口令	为单独的秘钥项指定口令。只有当用户签署 JAR 文件时用户才需要这个口令,在验证 JAR 文件时不需要。如果用户没有指定该命令选项,系统将提示用户输入它。出于对安全的考虑,一般在命令行中不指定这个口令 注意:keypass 口令可以与 keystore 口令相同。如果是这样,keypass 口令可以不指定
-signfile 文件	为.SF 文件和.DSA 文件指定基文件。如果没有指定,系统则使用别名的前 8 个字符(转换成大写字符)
-signedjar 文件	指定签署 JAR 后的文件的文件名(输出)。如果没有指定该值,则新的 JAR 文件将包含与它的源文件相同的名字(并且覆盖源文件)
-verify	指定用户文件中想要验证的签名。这基本上是与签署文件是相对应的 假设验证成功,系统将显示 jar verified 如果验证的是未签署过的 JAR 文件,或者使用的是签名系统不支持的算法(例如在未安装 RSA 提供者的情况下使用 RSA),那么系统将显示下列结果: jar is unsigned (signature missing or not parsable)
-ids	只有当 -verify 和 -verbose 选项同时使用时才可以使用 -ids 选项。当使用该选项时,系统将显示签名器的名字和 keystore 项的别名
-verbose idOrSigner	将 jarsigner 设置为冗余模式。在这种模式下,签名器输出签字过程和验证过程中的附加信息

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:在 JDK 1.1 以前,密钥工具的功能和 jarsigner 的功能是嵌入在 javakey 工具里面的。如果用户尚未升级到 1.2,仍然使用 javakey 的话,请注意密钥工具对 javakey 没有向下兼容的承诺,所以不能互换它们。

在使用 javakey 的过程中,当用户拥有公共密钥、私有密钥和证书时,还需要一些东西才能签署文件。这就是管理文件,管理文件指定了签名者、证书和签名文件使用的文件名。管理文件由名值对域组成,被要求的域在表 16.2 中给出。程序清单 16.4 是一个管理文件的示例。

表 16.2 要求的 JAR 管理文件域

域 名	域 值
signer	签名者的名字。这个名字必须已经在由 javakey 维护的永久数据库中注册过
cert	给定签名者使用的证书号。第一个证书为 1 号
chain	证书链的深度。这个域在目前的 JDK 版本中尚不支持;统统使用 0
signature.file	一个名字,8 个字符或更短,用于指定在签名的 JAR 档案的 META-INF 目录中的签名文件和证书文件

```
signer = liu
cert = 1
chain = 0
signature.file = LIUSIGN
```

为了签署 JAR 文件,可以使用带有-gs 选项和两个参数的 javakey 来实现,两个参数是管理文件的名字和 JAR 档案文件的名字。例如下面的命令使用管理文件 LiuJDF.txt 签署 Foo.jar:

```
javakey -gs Liu JDF.txt Foo.jar
```

作为对该命令的响应,javakey 在档案的 META - INF 目录中创建了两个项:签名文件 LIUSIGN.SF 和证书文件 LIUSIGN.DSA。

注意:虽然 JAR 档案某个特性被声称是能够签署单个文件,但是当前版本的 jarsigner 签字器看来尚未支持它。

16.4 java.util.zip 包

java.util.zip 包包含有许多操作 JAR 档案文件的类。虽然用户通常不需要使用这些类,但是在一般水平上理解它们还是有一定用途的。用户不必使用这些类去创建和加载 JAR 文

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

件;用户可以使用 jar 工具和 APPLET 标记来做这个工作。

java.util.zip 包定义了"检验和"接口,"检验和"接口为计算流的"检验和"的类定义了一个协议。java.util.zip 提供了两个实现"检验和"接口的类:Adler 32 和 CRC32。

16.4.1　类

java.util.zip 包定义了下列 14 个类:

ZipFile　zipFile 类描述了档案文件。它提供了读取文件项的方法。这个类不允许用户创建新的档案文件或修改已有文件的内容,创建和修改必须使用 jar 工具。

ZipEntry　ZipEntry 描述了档案文件的某个项,它拥有为该项读取和设置不同属性的方法,比如名字、修改时间和 CRC 检验和。另外,通过使用 ZipEntry 对象作为参数调用 ZipFile.getInputStream(),用户可以获得一个 InputStream 对象,该对象可以用于读取项的内容。

Adler32 和 CRC32　Adler32 类和 CRC32 类实现了"检验和"接口。它们计算一个数据流的两个不同的"检验和"。CRC-32 是一个工业标准算法;Adler-32 是一个由 ZLIB 作者之一 Mark Adler 使用类似的特性和较低的计算开销开发的"检验和"。为了使用这两个类,用户应该先将它们实例化,然后把它们作为参数传递给 CheckedInputStream 的构造器或 CheckedOutputStream 的构造器。事实上这正是 DeflaterOutputStream 和 InflaterInputStream 使用 Adler32 类要做的事情。

CheckedInputStream 和 CheckedOutputStream　CheckedInputStream 和 CheckedOutputStream 都扩展了 java.io.FilterOutputStream。它们维护着被读取和写入的数据的"检验和"。这些类中的每一个的构造器都使用流对象作为参数并有一个对象实现了"检验和"接口,"检验和"接口允许调用者为不同的流指定不同的"检验和"算法。

Deflater 和 Inflater　Deflater 和 Inflater 使用了标准的 deflater 算法实现了一般用途的压缩和解压缩。更详细的信息可以查看 RFC1951,RFC1951 可以从 http://www internic.ner/rfc/rfc1951.txt 上获得。

DeflaterOutputStream 和 InflaterInputStream　DeflaterOutputStream 和 InflaterInputStream 都扩展了 wa.io.Filter InputStream 和 java.io.FilterOutputStream。DeflaterOutputStream 压缩输出流;InflaterInputStream 解压缩输入流。这两个类建立了使用其它协议的其它压缩和解压缩的基础。这些协议包括 GZIP 协议(使用 GZIPOutputStream 和 GZIPInputStream)和 ZIP 协议(使用 ZIPOutputStream 和 ZIPInputStream)。

GZIPOutputStream 和 GZIPInputStream　GZIPOutputStream 和 GZIPInputStream 扩展了 DeflaterOut putStream 和 InflaterInputStream。它们使用标准的 GZIP 压缩算法压缩输出数据流和解压缩输入数据流。更详细的信息可查阅 RFC 1952,RFC 1952 可以从 http:// www . internic.net/rfc/rfc1951.txt 上获得。

ZIPOutputStream 和 ZIPInputStream　ZIPOutputStream 和 ZIPInputStream 扩展了 DeflaterOutputStream 和 InflaterInputStream。它们使用 Zip 压缩算法压缩输出流和解压缩输入流。

16.4.2　有计划地读取 JAR 文件

通常,用户不会使用 jva.util.zip 中的类去读取 JAR 文件;用户将在小应用程序标记中指定要阅读的档案,剩下的工作就由浏览器来做了。但是用户有时也需要自己来读取 JAR 文

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

件,本部分将介绍如何读取。

首先在名为 DumpJAR.java 的文件中输入程序清单 16.5 中所示的代码。

```
import java.util.zip.ZipFile;
import java.util.zip.ZipEntry;
import java.util.Enumeration;

class DumpJAR
{
    public static void main(String[] args)
    {
        String file_name = args[0];
        try
        {
            ZipFile zip = new ZipFile(file_name);
            PrintEntryNames(zip);
        }
        catch (java.io.IOException e)
        {
            System.out.println("Exception" + e);
        }
    }

    public static void PrintEntryNames(ZipFile zip)
    {
        for (Enumeration e = zip.entries(); e.hasMoreElements(); )
        {
            ZipEntry entry = (ZipEntry)e.nextElement();
            System.out.println(entry.getName());
        }
    }
}
```

现在让我们编译它:

```
D:/JDK 1.2/demo/TicTacToe/Jar > javac DumpJAR.java
```

如果用户在前面创建的 TicTacToe.Jar 文件上运行这个应用程序,用户将看到它的内容列表。注意项目的显示顺序与 jar 工具产生的顺序不一样。用户不要依赖这个由 ZipFile.entries()方法返回的项目顺序:

```
D:/JDK 1.2/TicTacToe/Jar > javac DumpJAR TicTacToe.jar
audio /return.an
```

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

audio/ding.au
TicTacToe.class
audio/yahool.au
audio/yahoo2.au
images/not.gif
audio/beep.au
images/cross.gif
META-INF/MANIFEST.MF

16.5　JAR 文件格式

JAR 文件格式是基于通用的和可以自由使用 ZLIB 文件的格式。ZLIB 格式是基于 ZIP 的文件格式。这是一个可移植的文件格式，它可以在某个目录层次上存储多个文件。ZLIB 格式不是针对某个特殊的压缩方法的格式，deflate 压缩方案是通用的。这是在 JAR 文件中使用的压缩方法。deflate 协议是基于 Lempee－Ziv 算法的变体 LZ77，它具有较低的压缩开销和适当的内存要求。这使得它成为十分通用的压缩协议。有关 ZLIB 更详细的信息，请查阅 RFC 1950 和 RFC 1951，用户可以从 ftp://ds.internic.net/rfc/上获得它们。

在一般情况下用户不需要关心 JAR 文件格式的细节，因为用户是通过 jar 工具和 javakey 工具或 java.util.zip 包与 JAR 文件进行交互。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 17 章　小应用程序和应用程序的比较

本章内容：

17.1　应用程序探究

17.2　应用程序的优点

17.3　开发 Java 应用程序

17.4　把小应用程序转换为应用程序

17.5　使用 Zip 文件格式将用户的应用程序打包

17.6　把应用程序转化成小应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

17.1　应用程序探究

虽然 Java 以其能够开发小应用程序而著名，实际上 Java 在开发各种其它应用程序方面也不失为一种强有力的语言。事实上 Java 能够被用来开发应用程序是 Java 强有力的特性。使用 Java 编写的应用程序没有传统的编程语言编程过程中出现的各种弊端。

一个几乎具有讽刺意味的事情是大多数人都忽略了 Java 具有创建应用程序的能力。当使用其它语言编写程序时，比如使用 C，C++ 或其它传统的语言，人们创建的总是标准程序。奇怪的是，围绕着小应用程序的宣传造成了一个奇怪的气氛，以致于人们完全忘记了 Java 除了能够创建小应用程序之外还可以创建应用程序。

应用程序和小应用程序之间的差别是深刻的，同时也是微妙的。事实上正如用户将在本章后面将学习到的那样，应用程序同时可以是小应用程序，小应用程序也可以同时是应用程序。小应用程序和应用程序之间最根本的不同是它们的工作环境。小应用程序必须"活"在像 Netscape Navigator，Microsoft Internet Explorer 和 AppletViewer 这样的浏览器中。而应用程序则可以在命令提示符下使用 Java 命令解释器（如果使用 JDK，解释器就是 java.exe）直接运行。

> **注意**：在将来，用户将可以不调用 Java 解释器从操作系统下直接运行 Java 应用程序。Microsoft，IBM 和 Apple 公司共同签署了一份协议，意在把 Java 虚拟机嵌入到他们的操作系统的未来版本中。另外，Sun 公司目前有一个代号为 Kona 的工程，它是一个完全基于 Java 的操作系统。当 Java 虚拟机成为操作系统的一部分时，Java 应用程序就会变得越来越重要了。

17.2　应用程序的优点

与小应用程序相比，应用程序模式提供了许多优点。其中之一就是应用程序运行速度可以更快。速度快是由两个因素引起的。首先，应用程序不需要使用浏览器。另外在运行小应用程序时，浏览器统一控制着小应用程序可使用的内存空间。作为应用程序，用户可以完全控制程序运行的整个环境。这些因素的结合导致了 Java 应用程序的执行略快于小应用程序，因为它的负担比小应用程序轻。

沙箱　应用程序和小应用程序之间的一个更重要的不同是应用程序不使用沙箱。沙箱限制了小应用程序的运行。在普通的环境里，小应用程序是禁止对本地文件系统做读取和写入操作的，并且小应用程序除了打开下载 HTML 文件和类文件的宿主的 URL 以外，不能打开国际互联网上的其它 URL。与小应用程序相比，应用程序就没有这些限制。当 Java 程序作为应用程序运行时，它拥有各种权力和能力，它与用其它语言，比如 C 语言编写的程序具有完全一样的权力和能力。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:如果用户读过第 16 章和第 34 章的内容,那么用户就知道如何创建签名小应用程序。这些小应用程序可以打开安全箱并允许小应用程序执行附加的操作。

另外,JDK 1.2 新的安全特性允许用户做正好相反的事情。用户可以对应用程序完全或部分地加以限制,使得它不再拥有比小应用程序更多的权力。

这意味着应用程序可以运行信任的方法。用户可以在 java.lang.RunTime 类中找到许多这种方法。当然也可以包含所有的本地方法和许多其它的东西。

假设用户不在意这些性能上的微小改善,也不需要访问沙箱以外的元素,那么为什么不把 AppletViewer(小应用程序浏览器)与小应用程序绑定在一起呢? 应用程序另外还有四个优点:

- 由应用程序产生的窗口不显示"Warning applet window "文本,它会对没有经验的用户造成困惑。
- 应用程序不需要 HTML 文件来告诉它们加载什么东西。
- 应用程序更加干净,因为它们的执行方式与普通的可执行程序一样。
- 无疑客户会认为应用程序是功能齐全的完整程序,并且从名字上理解,则认为小应用程序是装饰性程序。一般来说,这意味着客户们将更乐于为完整的程序付出金钱和时间。

17.3 开发 Java 应用程序

当用户在第 14 章"编写小应用程序"中学习编写 Java 小应用程序时首先学到的是任何小应用程序必须扩展 java.applet.Applet 类。与小应用程序不同,应用程序不需要为使自身可被使用而去扩展其它任何类。事实上,小应用程序需要扩展 java.applet.Applet 类的原因是因为应用程序(即浏览器)可以通过多态性使用这个类。

参见第 5 章"面向对象编程"的"通过继承扩充对象"部分

任何 Java 类都可以作为应用程序来运行。只有一个限制是:为了把类作为应用程序来运行,它必须拥有具有下列形式的 main 方法:

```
public static void main(String args[])
```

这样应用程序可以被认为是一个普通的类,这个类有一个特殊的特性:static public main 方法。在 Java 中,main()方法具有与 C 和C++ 中的 main()函数相同的用途;应用程序就是从 main()方法开始运行的。

17.3.1 HelloWorld 应用程序

正像我们在前面几章做过的和在本章后面将继续做的那样。让我们看一下功能低劣的"HelloWorld"程序,它是作为应用程序编写的,如程序清单 17.1 所示。

```
public class Hello{
    public static void main(String args[]){
        System.out.println("Hello World!");
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

用户可以用在本书中编译其它程序的方法编译 Hello 类。在命令提示符下输入：

```
javac Hello.java
```

如果是在 Macintosh 机器上就把 Hello.java 文件拖到 javac 图标上。

> **注意**：就和使用标准的公共类一样，Hello 类必须定义在以类名和.java 扩展名组成文件名的文件中。所以在本例中，必须把 Hello 类存储在名为 Hello.java 的文件名。

为了调用 Java 应用程序，用户将会使用[Java 类名]句法。注意用户只使用类名，不是“类名.class”或“类名.java”。Java 将搜索已有的类路径(这包含用户的当前目录[.]并试图找到用户指出的类。所以为了在命令提示符下运行 Hello 应用程序，可输入下列命令：

```
java Hello
```

这时用户可以在屏幕上看到 Hello World 信息。注意上面的命令行中没有输入 Hello.class，只是 Hello。Java 虚拟机隐含地知道 Hello 类就存放在 Hell.class 文件中，并且从类中的 main()方法开始运行。

> **注意**：在 Macintosh 机器上，事情有点不同，就像在我们学习编译 Java 程序时学到的那样。在运行 Java 应用程序时，需要双击 Java 图标然后输入用户希望运行的类的名字。另外也可以把小应用程序的类文件拖到 Java 图标上。
>
> 另外，对于 Windows 用户来说，要得到命令提示符，必须启动 MS-DOS 提示符窗口。

17.3.2　为应用程序传递参数

就像用户在“HelloWorld”应用程序中看到的那样，应用程序与小应用程序不同，小应用程序的加载不依赖于 HTML 文件。这有利于减少系统运行的复杂程度，但是如何为应用程序传递参数呢？

在 C/C++ 中，用户将典型地使用 argv 数组的值和 argc 的值。argv/argc 这种体系不太鲜明，许多程序员在每次使用这些变量时都要查找如何使用这些变量。在 Java 中，参数的设置是非常简单的。

用户可以回忆一下 main () 方法的原型，main () 方法有一个参数——一个字符串数组。这个字符串包含命令行中附加的参数的值。如果用户是 DOS 用户，可能会熟悉 /? 选项，例如：

```
dir /?
```

/? 对于 dir 程序来说是一个附加的参数。现在看一下如何使用 HelloWorld 程序做这件事情。我们不对全世界说 Hello，我们只对用户说 Hello。程序清单 17.2 显示了如何做这件事情。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 17.2 使用命令行参数的 HelloWorld

```
public class Hello{
    public static void main(String args[]){
        System.out.println("Hello" + args[0] + "!");
    }
}
```

为了编译该程序,请输入:

```
javac Hello.java
```

但是现在的 Hello 与以前的版本运行方法有一点不同,因为用户需要使用附加的参数:

```
java Hello Weber
```

现在用户可以看到的是:

```
Hello Weber
```

17.3.3 避免空指针异常

万一用户在命令行末尾中没有输入任何参数,那么用户将会看到:

```
java.lang.ArrayIndexOutOfBoundsException:
at Hell.main(Hello. Java):3
```

为了避免这条信息,即如用户忘记输入他(或她)的名字到命令行尾,用户应该事先在程序中设置错误检查机制。最好的办法是增加一个 if 语句。经修改后的 HelloWorld 程序如程序清单 17.3 所示。

程序清单 17.3 使用参数错误检查的 HelloWorld

```
public class Hello{
    public static void main(String args[]){
        if (args.length < 1){
            System.out.println("Syntax for running Hello is:");
            System.out.println( *          java Hello < Name > * );
            System.out.println(" \ n \ nWhere < Name > is the person to greet");
        } else
            System.out.println("Hello" + args[0] + "!");
    }
}
```

现在,如果用户在运行 HelloWorld 程序时没有输入任何参数,那么用户将会看到如下所示的信息:

```
Syntax for running Hello is
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
java Hello < Name>
```

这里 < Name > 是用户想问候的人的名字。

17.3.4　由 main( )的静态特性带来的限制

类的 main( )方法具有与 C 和C++ 的 main 函数非常相似的特性。但是又与 C 和C++ 的 main( )函数不同,因为 main 方法必须是静态的,它不能直接使用它的类的非静态方法和非静态变量,例如,用户无法编译下列代码:

```
  public class fooBar {
int foo ;
  public static void main (String args[]){
    foo = 50 ;
  }
}
```

当然,问题就在于 foo 变量不是静态变量,并且编译器拒绝静态的 main( )方法去修改它的值。为了理解为什么会是这样,让我们复习一下静态方法和静态变量的含义。当静态方法被加载到内存时,虚拟机看到它后会说:"好了,有一个这东西就行了,不管用户创建多少该类的实例,所以我就给它在内存中指定一个特殊位置,我对每个实例都使用它"。这事情不仅仅发生在第一次实例化时,而且发生在类被加载时。随后,当 fooBar 类被实际地实例化时,如果 main 方法被允许访问 foo 变量将会发生什么事情呢?

当 fooBar 类被实例化时,机器为 foo 变量分配空间,然后调用 main( )方法。且慢,我们知道,main( )方法已经被加载到内存中了,而且还引用了 foo 变量…,但是究竟引用的是哪一个 foo 变量呢? 当然我们现在是假设该类已经通过了编译器编译。所以说这个问题没有真实的答案,但是通过这个问题用户可以理解为什么编译器不允许用户执行这类操作。

用户可以通过两种途径来解决这个问题。第一个办法是把 foo 变量声明为静态变量,如程序清单 17.4 所示。

```
public class fooBar {
    static int foo;

    public init(){
        System.out.println("Init method");
    }

    public static void main(String args[]){
        foo = 50;
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

现在编译 fooBar 类，但是谁调用前面例子中的 init()方法呢？因为 init()方法不是静态的方法，编译器将再一次拒绝编译 fooBar 类。当然，用户可以把 init()方法也声明成静态的方法，但是这样做太麻烦了，而且作为一个程序员，如果用这种方法来完成一个较复杂的实用任务，就算不是不可能，也会感到非常困难。但是，如果在 main()方法中实例化 fooBar 的另一个拷贝可能是一个好主意，如程序清单 17.5 所示。

```
public class fooBar {
    int foo;

    public init(){
        System.out.println("Init method");
    }

    public static void main(String args[]){
        fooBar f = new fooBar();
        f.foo = 50;
        f.init();
    }
}
```

现在，因为 f 变量是在 main()方法中实际创建的实例，所以用户可以在 f 实例上执行操作。这里主要的不同是用户不是在 this 变量上执行操作，而是在 f.this 变量上执行操作，并且这种区别可以帮助编译器知道如何处理这些方法。换句话说，f 实际上是 fooBar 的实例。

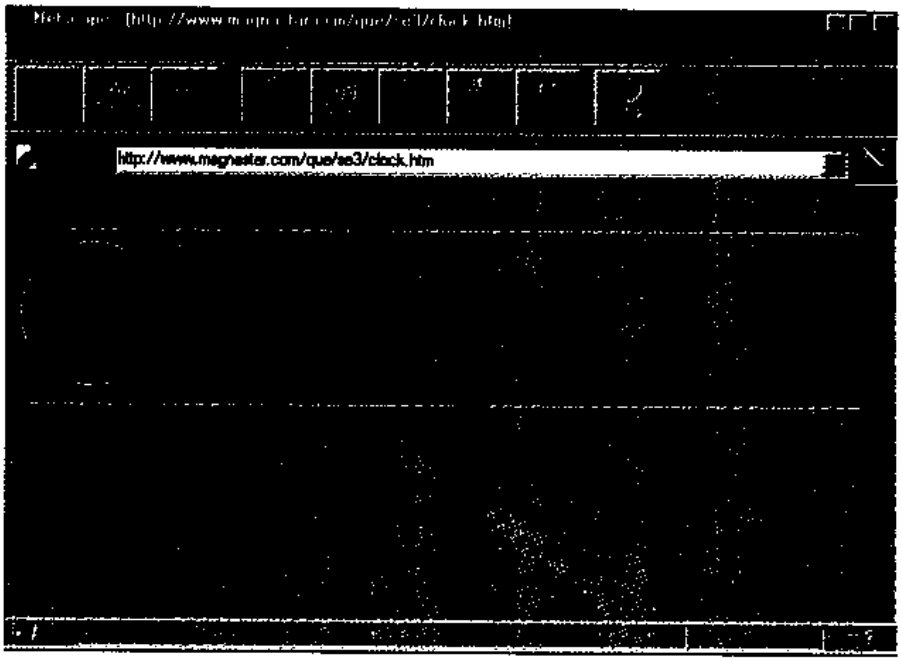


图 17.1 在 Navigator 中运行时钟小应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

17.4　把小应用程序转换为应用程序

我们已经大致地了解了如何创建应用程序，现在让我们考虑一下编写应用程序的另一个重要方面——将小应用程序转换为应用程序。用户已经知道，用户编写的小应用程序也可以当作应用程序来运行。在本部分中我们将为用户一步一步地演示如何把时钟小应用程序(如图 17.1 所示)转换为应用程序。

17.4.1　为什么要把小应用程序转换为应用程序

如果没有浏览器的帮助，小应用程序是不能够在命令提示符下运行的。

那么为什么要将小应用程序转换为应用程序呢？其实，信不信由你，在当今世界上并不是所有的人都知道国际互联网的存在。有的人，有的公司虽然知道，但是他们还没有访问国际互联网，还有一些公司连接了国际互联网，但是它们却不允许它们的用户访问国际互联网站点。更重要的事实是，许多人并不总是访问国际互联网站点。对于那些不常使用国际互联网的用户来说，网络上的应用程序就不是太有用了。

也许在不久的将来，用户也将把自己的小应用程序展示给不熟悉国际互联网的人们，或者把小应用程序展示给根本就不上国际互联网的人们。更有可能的是使用光盘传送小应用程序的示例。那么就没有理由让使用 Java 编写的应用程序显示与在国际互联网上显示的不一样的结果。设想一下若能够开发一个单一的应用程序不需修改任何代码，不需任何重新编译就可以在国际互联网上、企业网上和光盘上运行将是怎样一种意义。

17.4.2　把小应用程序代码修改成应用程序代码

在本章中，用户将把简单的时钟小应用程序修改成应用程序。程序清单 17.6 显示的就是时钟小应用程序的源代码。

```
/*
*
* Clock
*
*/
import java.applet.Applet;
import java.awt.*;
import java.util.*;

public class Clock extends Applet implements Runnable{
    Thread thisThread;

    Color faceColor,borderColor,minuteColor,hourColor,secondColor;

    public void init(){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        //read in the colors for each of the hands and for the face/border
        faceColor = readColor (getParameter("faceCol"));
        borderColor = readColor (getParameter("borderCol"));
        minuteColor = readColor (getParameter("minuteCol"));
        hourColor = readColor(getParameter("hourCol"));
        secondColor = readColor(getParameter("secondCol"));
    }

    // This method creates a color based on a string.
    // The string is assumed to be "red,green,blue" where each
    // of the colors is represented by its integer equivalent.
    public Color readColor(String aColor) {
      if (aColor == null) {
          return Color.black;
      }

      int r;
      int g;
      int b;

      //break the string apart into each number
      StringTokenizer st = new StringTokenizer(aColor, ",");

      try {
          r = Integer.valueOf(st.nextToken()).intValue();
          g = Integer.valueOf(st.nextToken()).intValue();
          b = Integer.valueOf(st.nextToken()).intValue();
          return new Color(r,g,b);
      }
      catch (Exception e) {
          System.out.println("An exception occurred trying to convert a parameter to a color:" + e);
          return Color.black;
      }
    }

    public void start(){
          thisThread = new Thread(this);
          thisThread.start();
    }

    public void run(){
          while(true){
              repaint();
              try{
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                thisThread.sleep(1000);
            }catch (Exception e){}
        }
    }

    public void update(Graphics g){
        paint(g);
    }

    public void paint(Graphics g){
        //fill clock face
        g.setColor(faceColor);
        g.fillOval(0,0,100,100);
        g.setColor(borderColor);
        g.drawOval(0,0,100,100);

        //get the current time
        Calendar d = Calendar.getInstance();
        //draw the minute hand
        g.setColor(minuteColor);
        double angle = (((double)(90 - d.get(Calendar.MINUTE)))/60) * 2 * Math.PI;
        g.drawLine(50,50,50 + (int)(Math.sin(angle) * 50),50 + (int)(Math.cos(angle) * 50));
        //draw the hour hand
        g.setColor(hourColor);
        angle = ((((double)18 - d.get(Calendar.HOUR_OF_DAY)
➡ + (double)d.get(Calendar.MINUTE)/60))/12) * 2 * Math.PI;
        g.drawLine(50,50,50 + (int)(Math.sin(angle) * 40),50 + (int)(Math.cos(angle) * 40));
        //draw the second hand
        g.setColor(secondColor);
        angle = (((double)(90 - d.get(Calendar.SECOND)))/60) * 2 * Math.PI;
        g.drawLine(50,50,50 + (int)(Math.sin(angle) * 50),50 + (int)(Math.cos(angle) * 50));
    }
}
```

首先，第一个任务就是在 Clock 类中增加 main()方法使得它成为一个应用程序。为了修改这个小应用程序，先用用户喜欢的编辑器打开 Clock.java 文件，向后翻页直到最后一个结束花括号(})。然后直接在该花括号之前加入程序清单 17.7 所示的代码。

```
static boolean inApplet = true;
public static void main(String args[]){
    /* set a boolean flag to show if you are in an applet or not */
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
inApplet = false;

/* Create a Frame to place our application in. */
/* You can change the string value to show your desired label */
/* for the frame */
Frame myFrame = new Frame ("Clock as an Application");

/* Create a clock instance. */
Clock myApp = new Clock();

/* Add the current application to the Frame */
myFrame.add (center*, myApp);

/* Resize the Frame to the desired size, and make it visible */
myFrame.setSize(100,130);
myFrame.show();

/* Run the methods the browser normally would */
myApp.init();
myApp.start();
}
```

在这里我们逐行地分析这个代码片段

```
inApplet = false;
```

该代码段的第一个语句创建了一个状态变量，通过这个变量用户可以区别该程序是以小应用程序运行的，还是以应用程序来运行的。正如用户将在后来了解到的那样，当用户不是在浏览器，比如 AppletViewer(小应用程序浏览器)或 Netscape(网景浏览器)中运行小应用程序时，用户必须做几件不同的事情，其中一件就在类中增加一个布尔变量(本例中是 inApplet)。从技术上讲，好的程序结构都是把对该变量的声明放在其它变量的最前面，但是现在放的位置也很容易看得清楚。注意该变量是被声明为静态的。如果用户漏掉了 static 这个关键字，编译器将因为用户在静态方法中引入了非静态的变量而拒绝编译。main()方法必须是静态的和公共的，这样用户才能够运行它。

```
Fram myFrame = new Frame("Clock as an Application");
```

接下来，用户创建一个用于放置新时钟的框架。参数 "Clock as an Application"被放置在框架的标题栏中。在标题栏中指明该程序正在按照应用程序方式运行是一个很好的做法；这种指示可以帮助部分用户消除疑惑。如果因为某种原因，用户不希望在构造器 Constructor 中设置标题，用户可以创建一个无标题框架(即 untitled)，以后再使用 SetTitle(String)把标题设置成用户希望使用的标题。

```
Clock myApp = new Clock( );
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接下来的一行是为 Clock 类创建一个新的实例。在这一点上有一个非常合理的问题是：为什么不使用 this 呢？毕竟 this 已经是 Clock 类的实例化了，对吗？我们创建新实例的主要原因是避免为了设置它们为静态的而重写小应用程序的任何方法。就像如果 inApplet 变量是非静态的变量，修改它的值是非法的一样，试图访问非静态的方法也是非法的，但是访问实例的静态变量是合法的。请记住，需要创建一个新的名为 myApp 的实例变量，并将它加入到框架中去。

```
myFrame.add("Center",myApp);
```

接下来的一行是在框架中加入一个新的时钟变量。这一点非常重要，因为在用户将时钟连接到某个东西上之前，时钟是不会被显示的。

参见第 21 章“容器和布局管理器”的“布局管理器”部分。

接下来用户将在 Clock.java 文件中加两行语句 myFrame.resize(100.130)和 myFrame.show()。myFrame.setSize(100,130)告诉应用程序将框架大小设置成 100×100 个像素点，但用户同时还要考虑在垂重直方向上，标题栏要占去 30 个像素点。一般来说，当用户将小应用程序转换为应用程序时，用户应该知道小应用程序的理想尺寸。当用户犹豫不决时，可以从最常用的 HTML 文件中直接拷贝出 WIDTH 值和 HEIGHT 值。一般来说这两个值不用修改就可以直接使用。但在少数情况下当用户希望修改这些值时，就可以使用在本章后面将要介绍的技术。在那里用户可以了解到如何计算参数数据和如何从命令行中读取尺寸。

```
myFrame.resize(100,100)
myFrame.show( );
```

警告：从技术上讲，当把小应用程序加入到一个框架中后，用户可以从一般的小应用程序的 init()方法和 start()方法开始运行该小应用程序，然而这种方法并不是理想的方法。如果用户的小应用程序采用了双缓冲或者要求使用 createImage(x,y)方法创建图像的话，那么直到框架被显示出来之前程序是不会继续工作的。缺点是当框架中什么东西没有时将会出现闪动。即使是用户不使用 createImage(x,y)现在也应该在脑海中记住这一点，因为这个小小的细节在别处没有被详细地记载，并且会引起作者数小时的头疼，因为它太容易被遗忘。

最后在用户的程序中增加两行 myApp.init( )和 myApp.start()。因为用户的应用程序不是运行在浏览器中的，所以 init()方法和 start()方法不会被自动地调用，如果是作为小应用程序运行，这两个方法就会被自动调用。所以用户必须显式地调用它们以模拟在浏览器中运行的效果。如果用户的应用程序没有出现在屏幕上，那么就需要在 main()方法的末尾加上一行 myApp.repaint()方法。

```
myApp.init( );
myApp.start( );
```

在存储用户的 Clock.java 应用程序的新拷贝之前，用户需要对它做一些修改。在程序的顶部检查是否引入了 java.awt.Frame。如果已经引入就可以存储文件到磁盘中了。

```
import java.awt.Frame
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

17.4.3 解释其它小应用程序的特性

在将小应用程序转换为应用程序的过程中，最困难的事情就是实现参数标记功能和其它小应用程序特有任务的功能。用户可以通过多种方式来解决这个问题，在接下来的部分里，我们将讨论两种最常用的解决办法。

17.4.4 缺省

第一种办法是缺省的。在缺省情况下，我们为应用程序中提供 HTML 文件能够提供给小应用程序的全部信息。在某种意义上讲，这就和用户使用 resize(x,y)指定框架的大小一样。为了实现 < param > 标记的项目的功能需要重写 getParameter()方法。

Clock 小应用程序需要从 < param > 标记中接收几个参数。现在让我们看一看 Clock 的 HTML 文件中这几个 < param > 标记，如程序清单 17.8 所示。

```
< TITLE > Clock < /TITLE >
< H1 > Clock  < /H1 >
< hr >
< applet code = "Clock.class" width = 100 height = 100 >
< param name = hourCol value = 255,00,00  >
< param name = minuteCol value = 00,255,00 >
< param name = secondCol value = 00,00,255 >
< param name = borderCol value = 255,255,255 >
< param name = faceCol value = 125,125,125 >
< /applet >
< hr >
```

为了在新的应用程序中模拟这些功能，我们需要在当前的应用程序 Clock.java 中增加如程序清单 17.9 所示的方法。

```
public String getParameter (String name){
    String ST;
    if (inApplet)
        return super.getParameter(name);
    //If you are not in an applet you default all of the values.
    if (name = = "hourCol")
        return "255,00,00";
    if (name = = "minuteCol")
        return "00,255,00";
    if (name = = "secondCol")
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            return "00,00,255";
        if (name == "borderCol")
            return "255,255,255";
        if (name == "faceCol")
            return "125,125,125";
        return null;
    }
```

警告:如果用户打算使用多个参数,那么用户就应该使用 switch 语句。switch 语句要求一个整数,用户可以使用字符串类的 hashCode()方法来获取这个整数。但是,因为多个字符串都可以拥有同一个 hashCode(),所以用户必须保证使用正确的字符串。这样就使得解决方案变得复杂了。如果用户要处理多个 < param > 标记就可以考虑使用接下来介绍的替代方法。

该方法使用用户自己的缺省值来替换一般由 java.applet.Applet 类实现的功能。

注意用户第一件事情是检查是否处在小应用程序中(if (inApplet))。如果是,那么就使用超类(java.applet.Applet)的 getParameter(String)方法。这样做可以使程序继续保留正常的运行方式,也就是说用户仍然可以把 Clock 当作小应用程序来运行。这样就可以使一个程序既可以作为小应用程序来运行,也可以做为应用程序来运行。

注意:处理 getParameter()更好的方法是实现 appletStub 模块。因为没有完整的接口说明,所以解释如何实现就太学术化了。如果用户打算实现 java.Applet.Applet 的多个方向的接口,可以从第 12 章"接口"中了解更多的信息。

17.4.4.1　重新编译应用程序

接下来的步骤是重新编译应用程序。重新编译应用程序的过程和编译小应用程序的过程没有什么不同的地方:

```
javac Clock.java
```

17.4.4.2　测试应用程序

现在用户就可以测试应用程序了(如图 17.2 所示)。为了测试应用程序,用户需要调用



图 17.2　作为应用程序运行的 Clock 程序

Java 虚拟机,然后再输入类名,如下所示:

```
java Clock
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

提示:时刻牢记要适当地使用大写字符。

17.4.5 加入 < param > 信息的第二种办法

应用程序用缺省方法取得 HTML 文件中的参数时显得既快捷又简便。但问题是,也许用户并不希望让这个参数固定不变,这样一来原来那种在程序的开始将参数值编写在程序里(即硬编码)的方法就不可取了。那么应用程序怎样才能从外界取得信息呢?最简单的答案是从命令行中读取。

正像用户可以回想起来的那样,main()方法可以从命令行中获取字符串数组来作为参数。用户也可以使用数组为运行时刻的应用程序传递信息。本部分将讲述一个最简单的例子:在命令行中向应用程序传递 WIDTH 和 HEIGHT 信息。虽然本部分也不解释如何插入 < param > 信息,但是希望用户自己能够从这个例子推演出插入 < param > 信息的做法。

为了能够使用命令行中的信息,用户需要在 mian( )方法中做一些修改。程序清单 17.10 显示了新的版本。

```
public static void main(String args[]){
    /* set a boolean flag to show if you are in an applet or not */
    inApplet = false;

    /* Create a Frame to place your application in. */
    /* You can change the string value to show your desired label */
    /* for the frame */
    Frame myFrame = new Frame ("Clock as an Application");

    /* Create a clock instance. */
    Clock myApp = new Clock();
    /* Add the current application to the Frame */
    myFrame.add ("Center",myApp);

    /* Resize the Frame to the desired size, and make it visible */
    /* Resize the Frame to the desired size, and make it visible */
    if (args.length >= 2)
      /* resize the Frame based on command line inputs */
      myFrame.setSize(Integer.parseInt(args[0]),Integer.parseInt(args[1]));
    else
      myFrame.setSize(100,130);
    myFrame.show();

    /* Run the methods the browser normally would */
    myApp.init();
    myApp.start();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
}
```

做完必要的修改后重新编译该程序。现在用户就可以运行任意显示尺寸的 Clock 应用程序了。用下面的命令试一下：

```
java Clock 100 100
```

乍一看这段代码，用户可能会感到这里的 main()方法与程序清单 17.3 中的 main()方法几乎是一样的。主要的差别在于下面这段代码：

```
/* Resize the Frame to the desired size ,and make it visible */
if(argv.length > = 2)
    /*  resize the Frame based on command line inputs */
    myFrame.setSize(Integer.parseInt(args[0]),Integer.parseInt (args[1]));
else
    myFrame.setSize(100,130);
```

实际代码的第一行检查用户是否在命令行中放置了足够的信息。检查的目的是避免因访问根本不存在的信息而导致的空指针异常。这样一来，即使用户在命令行中没有指明实际尺寸程序也能够正常地运行 Clock。

主要的工作是在接下来的语句行中完成的。对于用户来说这一语句行的结果是非常明显的，但是用户需要知道为什么要在数组值上使用 Integer.parseInt ()方法。在运行时刻 Java 虚拟机不关心命令行中带有什么；它看到的只是一个字符串。为了把字符串转换成整型数值，用户需要使用 Integer 类的 parseInt(String)方法。注意，使用的是 Integer 类，不是 int。如果用户仍感到困惑不解，那么请用户查阅第 7 章“数据类型及其它标志”。

警告：出于完整性考虑，用户应该使用 try{}块和 catch{}块将 parseInt()方法包括起来，以防止在命令行中输入的除整数以外的字符进入本段代码。

17.4.6　使用窗口来结束应用程序的工作

现在，用户也许注意到了，用户必须使用组合键 Ctrl + C 或其它方法来关闭应用程序。为了允许用户正常地关闭窗口，用户必须捕捉 WindowsClosing()事件的发生。用户也可以通过创建窗口监视器并在 WindowsClosing()方法中执行退出(exit)来实现对应用程序的关闭。

WindowListener 接口要求用户实现几个方法，所以这种做法并不总是最方便的做法。幸运的是，java.awt.event 包包含了几个方便的适配器，它们可以为用户处理这里面的细节。在使用 WindowListener 的过程中，WindowAdapter 类实现了接口并为每个方法提供了缺省行为。所以用户只要扩展 WindowAdapter 类并覆盖用户感兴趣的方法就可以了。

在 Clock 程序中，现在用户可以创建一个匿名的类并把 WindowAdapter 类加到 myFrame 变量中：

```
myFrame.addWindowListener(new WindowAdapter( ){
    public void windowClosing(WindowEvent event){
        System.exit(0);
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

});

最后，完整的 Clock 小应用程序的应用程序版本如程序清单 17.11 所示。

```
/*
 *
 * Clock
 *
 */
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class Clock extends Applet implements Runnable{
  Thread thisThread;

  Color faceColor ,borderColor,minuteColor,hourColor,secondColor;

  public void init(){
    //read in the colors for each of the hands and for the face/border
    faceColor = readColor (getParameter("faceCol"));
    borderColor = readColor (getParameter("borderCol"));
    minuteColor = readColor (getParameter("minuteCol"));
    hourColor = readColor(getParameter("hourCol"));
    secondColor = readColor(getParameter("secondCol"));
  }
  // This method creates a color based on a string.
  // The string is assumed to be "red,green,blue" where each
  // of the colors is represented by it's integer equivalent.
  public Color readColor(String aColor) {
    if (aColor == null) {
      return Color.black;
    }

    int r;
    int g;
    int b;

    //break the string apart into each number
    StringTokenizer st = new StringTokenizer(aColor, ",");

    try {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        r = Integer.valueOf(st.nextToken()).intValue();
        g = Integer.valueOf(st.nextToken()).intValue();
        b = Integer.valueOf(st.nextToken()).intValue();
        return new Color(r,g,b);
      }
      catch (Exception e) {
        System.out.println("A n exception occurred trying to convert a parameter to a color: " + e);
        return Color.black;
      }
    }

    public void start(){
      thisThread = new Thread(this);
      thisThread.start();
    }

    public void run(){
      while(true){
        repaint();
        try{
          thisThread.sleep(1000);
        }catch (Exception e){}
      }
    }

    public void update(Graphics g){
      paint(g);
    }

    public void paint(Graphics g){
      //fill clock face
      g.setColor(faceColor);
      g.fillOval(0,0,100,100);
      g.setColor(borderColor);
      g.drawOval(0,0,100,100);

      //get the current time
      Calendar d = Calendar.getInstance();
      //draw the minute hand
      g.setColor(minuteColor);
      double angle = (((double)(90 - d.get(Calendar.MINUTE)))/60) * 2 * Math.PI;
      g.drawLine(50,50,50+(int)(Math.sin(angle) * 50),50 + (int)(Math.cos(angle) * 50));
      //draw the hour hand
      g.setColor(hourColor);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    angle = ((((double)18 - d.get(Calendar.HOUR_OF_DAY) +
➡(double)d.get(Calendar.MINUTE)/60))/12) * 2 * Math.PI;
    g.drawLine(50,50,50 + (int)(Math.sin(angle) * 40),50 + (int)(Math.cos(angle) * 40));
    //draw the second hand
    g.setColor(secondColor);
    angle = (((double)(90 - d.get(Calendar.SECOND)))/60) * 2 * Math.PI;
    g.drawLine(50,50,50 + (int)(Math.sin(angle) * 50),50 + (int)(Math.cos(angle) * 50));
  }

  static boolean inApplet = true;
  public static void main(String args[]){
    /* set a boolean flag to show if you are in an applet or not */
    inApplet = false;

    /* Create a Frame to place your application in. */
    /* You can change the string value to show your desired label */
    /* for the frame */
    Frame myFrame = new Frame ("Clock as an Application");
    myFrame.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent event){
      System.exit(0);
      }
      });

    /* Create a clock instance. */
    Clock myApp = new Clock();

    /* Add the current application to the Frame */
    myFrame.add ("Center", myApp);

    /* Resize the Frame to the desired size, and make it visible */
    /* Resize the Frame to the desired size, and make it visible */
    if (args.length >= 2)
      /* resize the Frame based on command line inputs */
      myFrame.setSize(Integer.parseInt(args[0]),Integer.parseInt(args[1]));
    else
      myFrame.setSize(100,130);
    myFrame.show();
    /* Run the methods the browser normally would */
    myApp.init();
    myApp.start();
  }

  public String getParameter (String name){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        String ST;
        if (inApplet)
          return super.getParameter(name);
        //If you are not in an applet you default all of the values.
        if (name = = "hourCol")
          return "255,00,00";
        if (name = = "minuteCol")
          return "00,255,00";
        if (name = = "secondCol")
          return "00,00,255";
        if (name = = "borderCol")
          return "0,0,0";
        if (name = = "faceCol")
          return "125,125,125";
        return null;
      }

    }
```

让我们最后一次重新编译并运行 Clock 应用程序。现在如果用户单击窗口关闭图标，Clock 应用程序将像普通的程序一样从系统中退出。

17.4.7　检查所有的小应用程序的方法

当用户将自己的小应用程序转换成应用程序时，用户需要做的最后一步是找出由 java.applet.Applet 类提供的方法，这些方法只在浏览器中有效。一般地说来，用户需要在下面描述的这些部分中查找只在浏览器中有效的方法。

getAppletContext()　幸运的是用户使用 getAppletContext()方法做的大部分事情可以在应用程序中忽略掉。例如，如果脱离了浏览器，showDocument()就没有什么意义。当用户试图执行 getAppletContext().showDucument 时将会在 System.out 中产生错误，但是这个错误不至于导致应用程序崩溃。

类似地，showStatus()方法一般与应用程序也关系不大。在小应用程序中 showStatus 根据上下文显示有关的信息。最容易的解决办法是用 if(inApplet){}把这些代码括起来，使得当程序不以小应用程序运行时忽略这部分代码。

但是如果用户想看到这些信息应该怎么办？用户可以选择框架的顶 17 行和底 17 行把显示小应用程序上下文联系信息的 paint()方法写在这里。为什么要选择顶和底呢？由于框架对等的 UNIX 版本和框架对等的 Windows 95 版本之间的陌生的差异，每个系统辟出了一个 17 行的区域用于显示 Warning applet 信息。在 Windows 机器上这个区域是头 17 行；在 UNIX 机器上它是底 17 行。

如果用户不太信服上述解释可以在下述网站上查阅相关信息：

http://www.magnastar.com/ultra_nav

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

UltraNav 是一个由 MagnaStar 公司编写发行的程序。它是用于辅助站点导航的。注意 5 黄色的"information"行显示的位置取决于用户的平台。

如果用户使用是 Windows 机器,用户应该在框架的顶部看到信息栏。如果用户使用的是 UNIX 机器,信息栏在底部。不管信息栏是画在上面还是画在下面,用户只能看到一个。

getCodeBase()和 getDocumentBase() getCodeBase()和 getDocumentBase()方法的处理有点技巧。这两个方法都返回一个 URL,并且用户不要指望自己能使程序的使用者都与国际互联网连接。总之,如果程序的使用者能够询问用户的网站,用户就可能允许程序的使用者直接从用户机器上下载小应用程序,所以用户不需要把这样的小应用程序转换成应用程序。

用户通常需要根据各种情况来处理 getCodeBase()和 getDocumentBase()。如果用户可以不使用这些信息,那么就可以忽略它。如果用户确实需要使用由 getCodeBase()和 getDocumentBase()方法处理的信息,那么用户可能就需要硬编码 URL 或者从命令行中读取它。

注意构造器 在转换小应用程序时,经常需要用户自己创建实际的类构造器,不能使用空的构造器。创建自定义构造器可以使得从命令行或其它位置传递信息做得得心应手。如果用户创建了自定义构造器,那么一定不要忘记人工地再创建一个空构造器(空构造器是不使用任何输入参数的构造器)。如果用户创建了另一个构造器,Java 不会自动地为用户产生一个空构造器。用户不会注意到需要一个空构造器,当小应用程序工作时,如果另一个类需要创建该小应用程序的一个实例(因为多线索或其它什么原因),在这种情况下,这个类需要访问空构造器。即使用户不亲自从类中删除空构造器,空构造器也不会继续存在在那里。用户将会得到如下所示的错误信息:

```
java.lang.NoSuchMethodError
  at sun.applet.AppletPanel.run(AppletPanel.java:170)
  at java.lang.Thread(Thread.java)
```

这个错误信息并没有任何关于用户的类的信息。甚至说与用户的类没有任何牵连,它倒像是 AppletPanel 的一个错误(通常程序员称之为 bug)。如果用户遇到这种情况,第一件要做的事情是删除 *.class 然后再重新编译整个程序。这时编译器就能够捕捉到遗漏的构造器调用。

createImage 在用户把小应用程序转换为应用程序的过程中,如果用户使用 createImage 方法,而 Image 变量被返回一个空值,那么用户就首先应该修改框架的可视属性。具体做法请用户阅读本章前面的"将小应用程序代码修改成应用程序代码"部分。

17.5 使用 Zip 文件格式将用户的应用程序打包

现在用户已经把小应用程序转换成了应用程序,用户可以将它发送给自己的客户。发送应用程序的最佳方式就是使用 JAR 文件。为了将文件打包成 JAR 文件,用户可以按照第 16 章"JAR 档案文件。"里的指导操作。一旦生成了 JAR 文件,用户就需要为它增加类路径。这时就可以运行该程序了,就好像所有的类都已经在当前目录中一样。

17.6 把应用程序转换成小应用程序

将应用程序转换成小应用程序比起将小应用程序转换成应用程序的复杂程度小多了。能

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

够将应用程序转化成小应用程序的部分就是获取应用程序运行的基本功能。做到这一点只需要有一个设计决策。问题是当使用者离开用户的网页时用户是否希望启动或停止用户的应用程序？或者当使用者离开用户的网页时用户是否希望启动一次应用程序？

17.6.1　最简单的转换

假设用户希望启动一次应用程序，那么用户需要做的事情是扩展 java.applet.Applet 和在 init()方法中调用 main 方法：

```
public void init( ){
main(null);
}
```

注意：如果用户已经扩展了另一个类应该怎么办？唉，真是不幸，这个问题没有什么好的答案，除非用户扩展的是 Panel 类、Container 类和 Component 类。如果是上述三种情况之一，那么就可以使用 Applet 类来代替原来的类（也就是把原来扩展的类改成扩展 Applet）。多态性将虚拟地保持应用程序不发生变动。如果用户没有这么幸运的话，用户就需寻找另外的方法去扩展 Applet 类了。两种常被扩展的类可能能够解决这个问题，但是要根据具体情况来定。

Thread——修改用户的应用程序去实现 Runnable。如果用户需要访问 Thread 方法，用户将需要为运行 Thread 方法而保留一个实例变量。

Frame——取决于用户使用 Frame 做什么，用户可能能够扩展 Applet，但是偶尔也需要创建一个 Frame 实例并把 Applet 加进实例中：

```
Frame fr = new Frame( );
fr.add(this);
```

所有框架调用都将需要被指向 fr 变量。

17.6.2　处理命令行参数

如果用户接受了命令行参数，用户当然可以用设置 < PARAM > 标记来解释这些参数或传递到一个数组中，然后用户再将它们发送给应用程序。如下所示：

```
public void init( ){
String args [ ] = new String[2];
args[1] = getParameter("param1")
args[2] = getParameter("param2")
main(args);
}
```

17.6.3　维护应用程序的一个单一实例

如果不是启动一次应用程序而是让它没有限制，那么用户可能希望应用程序在用户进入和离开网页时启动和停止，用户应该考虑一下这该怎么处理。最容易的解决办法是通过把方法修改成 start()，而不是 init()，每次在 start()方法中创建一个应用程序的实例来实现。在

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

stop()方法中,用户可以执行正常的应用程序的退出过程。

另一方面,如果用户希望离开应用程序,但仅仅是希望应用程序进入休眠状态,那么用户还必须做一些其它工作。究竟该如何来做,完全取决于用户的程序。

如果应用程序是在 Frame()中,那么用户就可以在接下来的实例中通过隐藏(使用 hide()方法)和显示(使用 show()方法)来实现这一目的:

```
public void init( ){
String args[ ] = new String[2];
args[1] = getParameter("param1");
args[2] = getParameter("param2");
main(args);
}
public void start ( ){
myApp.show( );
}
public void stop( ){
myApp.hide( );
}
```

为了使它能够工作,用户需要保留在 mian()方法中产生的 myApp 的一个实例方法。

17.6.4 在应用程序向小应用程序转换过程中更难的问题

应用程序向小应用程序转换过程中一个非常难以处理的问题是:在小应用程序中恢复安全保护方法。遗憾的是有些问题没有答案。例如 System.exec()就不能在小应用程序中调用。

最常见的问题是文件访问问题,遗憾的是没有真正的办法能够完全恢复对文件的访问,尤其是对文件的读取和写入。有一个可能的办法是实际存在的,即如果用户需要做的事情是从文件中读取 URL 类,这是可以实现的。用户可以看下面的两个代码片段,它们能够产生可以读写文件的 InputStreams(is 变量)。

```
try{
FileInputStreams fs = new FileInputStream("myFile.txt")
InputStreams is = new InputStream(fs);
}catch{ ~
try{
URL us = new URL(http:"//www.magnastar.com/myFile. txt");
InputStreams is = us .openStream( );
}catch{ ~
```

这里有一点不同是 myFile.txt 是位于不同位置的。在第一种情况下,它位于本地文件系统中;在后一种情况下,它位于网络服务器中,对于配置文件来说,这可能就可以了。

如果用户坚持要写一个文件,那么用户需要或者签署该小应用程序并限制在支持本地文件 1.1 版本浏览器(在作者编写这一段内容时,只有 HotJava 能够支持)上运行;或者编写一个客户—服务器方式的应用程序并在服务器端把信息存储到文件中。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

第 18 章　管理应用程序

本章内容：

18.1　安装应用程序

18.2　在同一个系统中维护多个应用程序

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

18.1 安装应用程序

Java 应用程序是传送 Java 程序强有力的手段。在用户可以使用 Java 应用程序之前，用户必须知道如何安装它们。本章将讨论如何安装和维护应用程序。本章也提供小应用程序向应用程序转换的指导。

Java 应用程序有许多形式，但是本章只讨论最普通的两种：被打包成一系列 .class 文件的应用程序和单一的 zip 形式的应用程序。

警告：当用户安装别人的 Java 应用程序时，用户实际上放弃了使用小应用程序时得到的安全保护。放弃安全保护并不见得是一件坏事；事实上，用户可能需要违反这种安全保护。但是用户应该意识到安装随机的 Java 应用程序可能会为用户带来各式各样的问题，这和用户在使用传统的软件中遇到的情况是一样的，都有可能遇到病毒和其它的恶意软件的危胁。

18.1.1 从 .class 文件中安装应用程序

从一套 .class 文件中安装应用程序比起从 .zip 文件中安装应用程序稍微简单一些。将来，大部分应用程序都会有自己的安装程序，但是现在用户还必须手工地执行安装，在接下来的部分里我们将解释如何安装时钟应用程序，时钟应用程序包含在本书附带的光盘上。我们在上一章中讲解了该时钟应用程序（即 Clock）。

为应用程序创建目录 首先，用户需要创建一个目录来存放时钟应用程序。这个目录不需要与存放 Java JDK 的目录关联；但是有一个应用程序存放计划还是很重要的。这个计划可以与用户安装传统程序的计划是同一个，比如 Netscape，或者是对于用户的 Java 应用程序来说是唯一的东西。

警告：就像使用任何其它程序和数据时一样，对于用户认为是重要的 Java 应用程序也需要进行备份。尽管应用程序不会轻意崩溃，但还是不要忽略这种可能性。

拷贝文件 当用户创建完用于存放应用程序的目录后，就可以把所存的 .class 文件拷贝到该目录下。用户应该确保在用户维护的目录结构中已经为该应用程序设置了目录，如果用户有为包开设的子目录，应该确保这些类也在子目录中。

注意：在 Windows 机器上的 DOS 提示符下拷贝整个子目录的命令是：

```
xcopy c: \ original \ directory \ * .class c: \ destination \ directory/s
```

用户也可以在 Windows 的浏览器系统中拖放某个目录结构。

在 UNIX 机器上拷贝目录的命令是：

```
cp -r /orignal/directory /destination/directory
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

警告：如果用户正在对应用程序进行配置，或者用户正升级该程序，那么不要让工作拷贝被使用者访问。如果在用户正在编译应用程序时其它人启动了它，那么就可能发生一些不可预料的严重后果。

确保一切正常　现在应该确保每一件事情都按照它应该的方式去运行。进入存放时钟应用程序存放的目录并输入 java Clock 命令。如果一切都在照计划执行，那么屏幕上将会出现时钟窗口，如图 18.1 所示。如果没有出现，那么说明某个地方出现错误了。检查一下每个步骤看是否每一步都正确。看一看 Java 在当前目录中能否运行得起来。如果用户正确地安装了 Java 开发工具，那么在当前的类路径中就已经包含有可执行的 Java 程序(java.exe)；如果不是这样，请用户查阅第 3 章“安装并启动 JDK”中的有关内容。

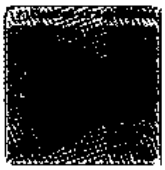


图 18.1　在 Solaris 下运行的时钟程序

18.1.2　结束安装

当正确地拷贝所有的.class 文件之后，接下来的任务是创建用于运行应用程序的描述文件或批处理文件。

用户可以使程序的运行自动化，这样就不需要每次都输入 java Clock。如果只输入 Clock 就能调用 Clock 程序，而用不着连 java 都输入进去，那么 Clock 程序的使用者会感到非常高兴。

用户做到这一点，在各个平台上的做法不太一样，在这里我们只介绍如何在 UNIX 系统和 Windows 95/NT 上实现方法。

18.1.3　结束 UNIX 操作系统下的应用程序安装

在解释如何在 UNIX 系统下安装应用程序时，我们着重介绍如何在 Solaris 2.4 下使用 Korn Shell 安装应用程序。与用户的实现过程可能有一些轻微的差别，这取决于用户特定的操作系统和 shell。

创建封套脚本　在 UNIX 环境下使 Java 应用程序的用法自动化是通过创建封套脚本来实现的。封套脚本是把所有用于 Java 应用程序的命令都“封套”在一起的脚本文件。第一个任务是创建脚本。用户可以使用 vi，nedit 或用户喜欢的文本编辑器来创建这个脚本。程序清单 18.1 显示了一个 clock 脚本的示例。注意这里有几个变量用户需要针对用户具体的安装做一些修改。

```
# Add the applications directory to the CLASSPATH
# set to the directory you have placed the application
# Note, I insert the application directory first to avoid
# having classes from other applications getting called first
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CLASSPATH = /ns - home/docs/que/Clock/: $ CLASSPATH

# Set the location in which you hold java.
# This directory is probably the same as below
# If you have java in your global path, this line is
# not really necessary

Java _ Home = /optl/java/bin/java

# Specify the name of the application.
# Important: Remember this is the name of the class, not the
# file

App = Clock

# Now run the actual program.
# If you have any additional parameters which you need to
# pass to the application, you can add them here.

$ Java _ Home $ App
```

测试脚本 将程序清单 18.1 中的文本拷贝到名为 Clock 的文件中,并且要确保脚本的功能正确。为了对它进行测试,可以简单地输入封套脚本的名字,如下所示:

```
Clock &
```

这时,用户的应用程序应该开始启动,显示结果如图 18.2 所示。如果不能正常运行,检查一下脚本文件是否是可执行的。用户可以通过输入下列命令来使得脚本成为可执行的文件:

```
chmod a + x Clock
```

如果用户不希望每个人都能执行用户的脚本则不要做这项工作。如果是这样,可输入 chmod u + x Clock 或者询问系统管理员 chmod 应该使用哪些适当的参数。

将脚本拷贝到公共的地方 用户可以考虑把新的脚本文件放置在/user/bin 目录下,这样做可以使得所有能够访问系统的人都能运行新的脚本。

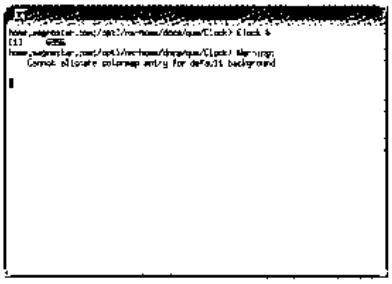


图 18.2 测试时钟应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

18.1.4　结束 Windows 操作系统下的应用程序安装

本部分将讨论如何在 Windows 95 操作系统下如何安装应用程序。除了几个个别的地方有些不同外，这个过程与在 Windows NT 操作系统下的安装过程是一样的。

用户可以通过两种途径实现在 Windows 操作系统下的安装：创建批处理文件和使用 .pif 文件。

创建封套批处理文件。为了使用批处理文件安装应用程序，必须首先创建批处理文件。在创建批处理文件时，用户可以使用用户喜欢的编辑器。用户可以使用 DOS 提供的 Edit 命令，Windows 的记事本或其它的文本编辑器。使用文本编辑器创建一个名为 Clock.bat 的批处理文件，该文件中包含如程序清单 18.2 所示的内容。

提示：如果用户的应用程序不使用 Windows 环境，或者用户需要能够在 System.out 中看到输出，那么就应该用 java 替换 javaw。

```
rem add the location where java.exe is located. If it
rem is already in your path don't add this line.
rem change c:\java\bin to the directory you have
rem installed for the JDK - see chapter 3

set PATH = %PATH%;c:\java\bin

rem Set this line to be the directory where your new
rem application is located.

set CLASSPATH = c:\appdir\;%CLASSPATH%

rem Run the actual application, change the applClass to be
rem the correct class for the application you are installing

javaw applClass
```

注意：如果用户的应用程序不能运行，并且显示类似于“Can't find class classname”这样的错误，用户首先要检查 .zip 文件所在的目录是否包含在 CLASSPATH 变量指定的那些目录中。其次则需要检查 CLASSPATH 变量中的字节数是否超过了最大限制，这个限制在 Windows 操作系统下是 128 个字符。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

警告:与 Solaris 和 Macintosh 机器不同,Windows 机器上的 CLASSPATH 不是使用冒号(:)分隔的,CLASSPATH 中的元素是使用分号(;)来分隔的,简单地说 CLASSPATH 的语法与设置 PATH 变量的语法是相同的。

错误的语法:

```
set CLASSPATH=C:\ java\jre\lib\rt.jar;c:\application\
```

正确的语法:

```
CLASSPATH=c:\java\jre\lib\rt.jar;c:\application\
```

从技术上讲在 JDK 1.2 中,每个选择都可以工作,但是对于仍然使用 JDK 1.0 的用户来说,不要指望这个功能是向下兼容的。

> **注意**:如果用户正在使用的是 1.1 版本的 JDK,那么用户必须在本章中把所有的 jre\lib\rt.jar 的全部实例用 lib\classes.zip 来替换掉。

测试批处理文件 为了运行应用程序,用户可以在 DOS 提示符下输入 Clock 命令。这时应用程序应该开始启动,并显示如图 18.3 所示的结果。用户应该特别注意向应用程序传递的几个附加参数。因为 Clock.bat 是一个 DOS 批处理文件,所以它对大小写是不敏感的,也就是说使用 Clock , clock 和 cLOCK 的效果是一样的。

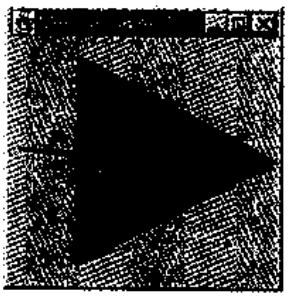


图 18.3 运行在 Windows 系统下的时钟应用程序

在 Windows 操作系统下添加应用程序 为了把这个批处理文件加进 Windows 环境中,让我们切换回 Windows 环境,如果有必要,先选择存放应用程序的文件夹为当前文件夹。应确保能够看到该文件夹中的内容,而不仅仅是能够看到该文件夹的图标。

现在让我们创建一个新的捷径(选择 File 菜单,再选择 New 选项中的 Shortcut 选项)。填写启动新的批处理文件的信息(在本例中是 c:\que\Clock.bat),如图 18.4 所示,然后指定在桌面上出现时应用程序使用的名字。

当用户创建完捷径以后,可以用鼠标双击它。这时 MS-DOS 窗口被打开,并且时钟程序也开始启动。现在如果用户像大多数人那样讨厌弹出一个 DOS 窗口,一般情况下当用户在看到 System.out 的输出时也看到一个大黑块,往往会将 DOS 窗口关闭掉。下面是几种消除这个讨厌的显示结果的办法。

为了使 DOS 窗口不那么讨厌,首先要制止 DOS 窗口出现在屏幕上,其次当 Java 应用程序开始启动后要使 DOS 窗口自动退出系统。为了实现这些修改,需要为新的应用程序打开属性窗口。具体操作是将鼠标光标移动到时钟图标上,然后用鼠标右键单击时钟图标。这时有--

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

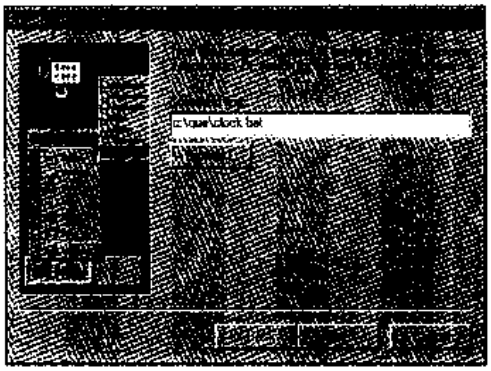


图 18.4　捷径窗口

个弹出式菜单出现在屏幕上;选择其中的 Properties 选项打开属性窗口,如图 18.5 所示。现在切换到 Program 表页上。将 Run 选项的值修改成 Minimized;这样就可以使得 DOS 窗口不再出现了。接下来将 Close on Exit 复选框选中,这个选项值可以使 DOS 进程在 Java 应用程序启动之后自动退出系统。最后单击 OK 按钮。

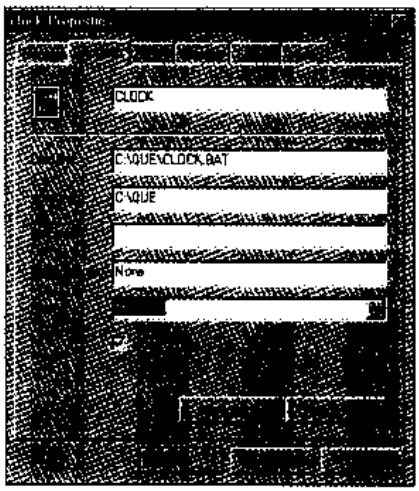


图 18.5　在属性窗口中修改 Run 选项并设置 Close on Exit 复选框

现在,如果用户双击时钟应用程序的图标,时钟应用程序就不再显示讨厌的 DOS 窗口了。

另一个向 Windows 操作系统添加应用程序的方法要求用户知道下列情况:

- java.exe 文件在什么地方,java.exe 文件必须位于命令路径之中。
- rt.jar 文件在什么地方。
- 新的应用程序在什么地方。

首先选择新的应用程序所在的文件夹。然后按照前面介绍的方法创建一个捷径。当系统提示用户输入命令行选项时,输入下面这行信息,如图 18.6 所示。

```
c: \ java \ bin \ javaw.exe  - classpath c: \ java \ jre \ lib \ rt.jar;c: \ appDir \ applicationClass
```

在用户的输入中,可以将路径名和类名全部替换成用户的应用程序所在路径和所用的类名。如果用户的应用程序未被加载,可以使用 java.exe 代替 javaw.exe。javaw.exe 是 java.exe 的替代版本,javaw.exe 在应用程序启动以后立即返回,并忽略应用程序启动过程中产生的所有错误信息。如果应用程序的使用者不关心应用程序工作的内部细节,那么使用 javaw.exe 是很方便的。但是当出现错误时,使用者很难发现问题。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

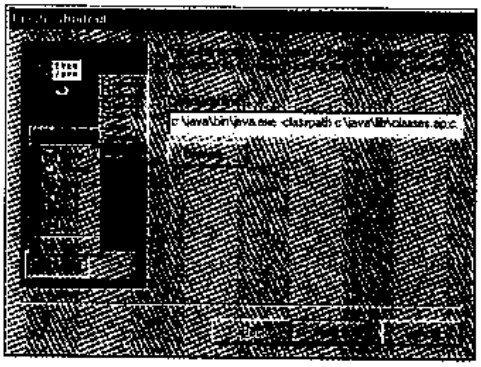


图 18.6 在捷径窗口中输入完整的命令行

18.1.5 从.jar 文件中安装应用程序

从 1.1 版本开始增加了对使用单个.jar 文件发布 Java 应用程序的支持。这一增加使得发布应用程序变得更加简单,因为这样只需要.jar 文件和封套就可以了。

当用户从.jar 文件中安装应用程序时,应该注意下列事项:

- 当从.jar 文件中运行应用程序时,用户必须把该文件包含在 CLASSPATH 环境变量中。用户必须实际地指明.jar 文件,不能只指明存放它的路径。
- 当用户在 CLASSPATH 变量中包含多个小应用程序时,应确保正在安装的应用程序不使用与在这些小应用程序中引用的类同名的类。

第二点很重要。假设用户已经把一个名叫 SkyTune 的小应用程序安装在了 CLASSPATH 中,SkyToon 是用于计算今天降水概率的程序,SkyToon 中有一个类名叫 Tune,它是处理天空颜色的。另外用户还有一个应用程序名为 CDTunes,用户使用这个小应用程序可以播放光盘驱动器里的音乐 CD。CDTune 有一个类名叫 Tune,它是用于处理 CD 上音频的输入和输出的。在这种情况下,两个应用程序中都有一个名叫 Tune 的类,那么 Java 将会怎样调用呢? 当用户运行 CDTunes 时,Java 解释器在类路径中开始查找,当找到 Tune 类的第一个实例时,找到的是 SkyToons 应用程序的类,那么结果只能是导致死机。

用户可以通过精心命名或分类打包来避免这种情况的发生。当然最好的解决办法是使用命名妥善的包。如果用户运行别人编写的程序,那么就很难保证这类问题不会发生了。用户应该意识到应用程序会因此而在某一天停止工作(在运行以.class 形式存放的应用程序时也会遇到这种问题,但这种问题在出现时比较明显。)

现在用户可以开始安装用.jar 格式发布的应用程序了。虽然从管理的角度来说,用.jar 文件发布的类比用.class 形式发布的类更难处理一些,但是用户必须对封套脚本或批处理文件所做的修改只有一个。

如果用户使用的是 UNIX 操作系统,可以引用程序清单 18.1 中的脚本并将下述语句

```
CLASSPATH="/ns-home/docs/appDir/:$CLASSPATH"
```

修改成:

```
CLASSPATH="/ns-home/docs/appDir/Application.zip:$CLASSPATH"
```

如果用户使用的是 Windows 操作系统,可以使用程序清单 18.2 中的批处理文件并把

```
set CLASSPATH=c:\appdir\;%CLASSPATH%
```

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

修改成

```
set CLASSPATH = c: \ appdir \ application.jar; %CLASSPATH%
```

这些例子假设用户接收的应用程序的.jar 文件的文件名叫 application.jar。实际情况下，这个文件可能是 classes.jar。这些例子只是用来说明文件可以使用任何名字。

18.2　在同一个系统中维护多个应用程序

在单个系统中维护多个 Java 应用程序不像维护几个已经编译成二进制码的程序那么简单。原因在于：

- Java 字节码依赖于虚拟机(可查阅第 1 章“Java 能做什么”)。因此，虚拟机的改变可能会导致用户的所有应用程序出现新错误和老错误的消失。
- Java 程序不能被编译成一个单一的文件。程序中的每个类都被包含在各包的文件中。代码是基于它的类名安装的。
- Java 应用程序重新使用了其它的程序，如果这些其它程序被修改过了也会影响 Java 应用程序。

用户可以自己解决后两个问题。但是第一个问题必须由虚拟机销售商来解决。

让我们考虑一下一个示例情况。用户已经使用了几个月基于 Java 字处理器；某一天，用户或用户的系统管理员安装了一个新版本的 Java(.exe)。这个 Java 新版本比现在的版本的速度快 300 倍(这只是下一代虚拟机速度的极小值)，但是它有一个很有意思的副作用：它对字符 a 和字符 z 作切换。这种切换可能是由错误导致的。但是字处理器该怎么办？更糟糕的是，如果用户使用了好几天也没有发觉，并且将原来的某些文档用新的格式存储了，那该怎么办？只有一种办法可以完全防止这种事情的发生，那就是在证实新版本与旧版本 100% 兼容之前不要对虚拟机进行升级。那些在 Java 最终版本发生之前使用 Java 开发了程序的开发者们，在每个新的 JDK 版本发行时都要经历“成长的”痛苦(从 JDK pre-beta 升级到 JDK beta1，等等)。不过用户不要因此而害怕升级虚拟机。其实大多数问题都不是太大的；但它们几乎都要求修改少量的代码。并且如果没有源代码，那么就可以不做这种升级修改工作了。

第二个问题是 Java 编译后的文件名都是 xxx.class，每一个小应用程序的类都包含在自己的.class 文件中，并且每个应用程序都可以包含数十个类文件。

对于用户的每个包含有数十个类的应用程序来说，很难避免类名的冲突，一旦发生这种情况，就可能导致加载错误。一个解决的办法是把类放进包里，并为每个包起一个唯一的名字。那么当包重名了该怎么办？如果用户有良好的编程习惯和经验，那么这种情况就不会发生。万一发生这种情况，我只能说：这个世界并非是完美无缺的。

为了防止这个问题导致应用程序崩溃，用户应该总是把当前的应用程序目录或.jar 文件放在类路径表的开始处。在有代码共享的情况下，应确保包含在 CLASSPATH 中的条目对于正在运行的应用程序是正确的。如果用户认为自己正在使用一个错误的类，那么可以先把 CLASSPAH 清空，然后只使用需要的目录来建立它。最后用户应该相信程序的开发者，不会导致类名冲突。随着虚拟机的更新，到时候会开发出来好的方法可以避免这种情况的发生。

最后，用户如何处理与其它应用程序共享的代码和从国际互联网上下载来的别人的代码？当用户修改某个应用程序时，用户必须保证这种修改是向下兼容的。一般情况下，共享的代码

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

不会经常修改。但是，当用户安装应用程序的新的版本时，用户应当确保其它程序都不依赖原来旧版本中的代码。

如果某些程序依赖老版本中的代码，为了维护合法性，用户需要重新安装其它应用程序需要的那部分代码。

总之，这个过程不会像安装 Microsoft Word 的新版本那么简单，但是也不会像重新安装操作系统那么复杂。安装 Java 应用程序的关键是需要注意由每个修改可能引起的连锁问题。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第三部分　用户界面

第 19 章　java.awt：部件
第 20 章　深入研究异常和事件
第 21 章　容器和布局管理器
第 22 章　图形
第 23 章　JFC——Java 基础类
第 24 章　高级 JFC
第 25 章　图像
第 26 章　Java 二维图形

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 19 章　java.awt:部件

本章内容:

19.1　使用 java.awt 构造图形用户界面
19.2　按钮
19.3　按照 1.1 版本的事件模式使用按钮
19.4　标签
19.5　复选框和单选按钮
19.6　选择框
19.7　列表框
19.8　文本域和文本区
19.9　滚动条
19.10　画布
19.11　部件的公共方法

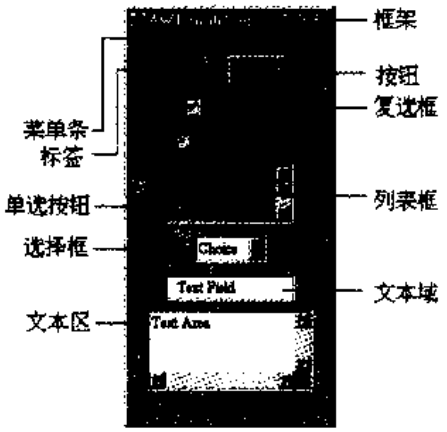
Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

19.1 使用 java.awt 构造图形用户界面

部件是 AWT 的构造块。最终的用户界面直接与这些部件交互。AWT 包提供的部件有：

- 按钮
- 文本域
- 标签
- 文本区
- 复选框
- 菜单
- 单选按钮
- 画布
- 列表框
- 滚动条
- 选择框

图 19.1 是一个使用了部件的小应用程序示例，该小应用程序使用了 AWT 包中的一些部件。



19.1 AWT 主要是由一些大家熟悉的部件组成

图 19.2 显示了部分 AWT 部件的继承层次。

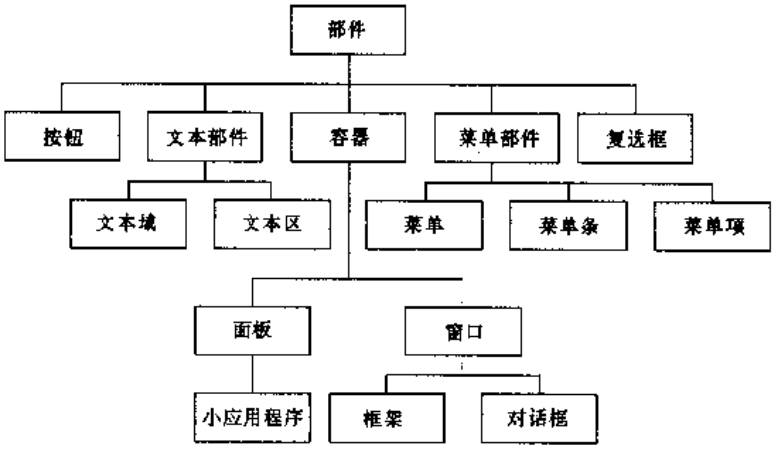


图 19.2 AWT 继承了 Component 类的所有用户界面部件

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

19.2　按钮

尽管按钮的机制非常简单,但是在许多图形界面中都大量地使用它。用户在工具条、对话框、窗口、甚至滚动条等部件中都可以看到它们。

19.2.1　创建按钮

在创建按钮时用户唯一需要做的决策是是否给按钮填写标签。除此之外按钮没有其它的选项。

为了创建无标签按钮,用户可以使用空的构造器。

```
public Button()
```

创建有标签按钮也一样简单:

```
public Button(String label)
```

在创建按钮之后,需要把按钮加进容器内。因为小应用程序已经是一个容器,所以用户可以把按钮直接加进小应用程序:

```
Button myBotton = new Button("Press Me");
add(myButton);
```

要想改变按钮的标签,可以使用 setLabel 方法来实现:

```
public void setLabel(String newLabel)
```

要想读取按钮的标签,可以使用 getLabel 方法来实现:

```
public String getLabel( )
```

注意:用户可能注意到了这里没有图像按钮,即按钮包含的是图像标签而不是文本标签,这种按钮在创建工具条时几乎是必须的。遗憾的是在 AWT 中不支持这种按钮。如果用户希望使用图像按钮,那么用户必须自己来实现它,或者从国际互联网上的 Java 档案中寻找别人实现的图像按钮。另外,如果用户选择使用 Java 基础类(在第 23 章和 24 章中将会介绍),那么可以使用 JButton 类来实现图像按钮。

19.2.2　使用按钮

既然用户已经学会了在小应用程序中创建按钮,那么现在让我们继续介绍如何使用按钮来做一些事情。处理 Java 中的事件有两种办法。在 Java 1.0 事件模式中,所有的部件都有一个名为 action 的方法,当部件上面有动作发生时,系统就调用 action 方法。在按钮事件中,action 方法是在按钮被按下时得到调用的。按钮的 action 方法是类似于 keyDown 和 mouseDown 方法的事件处理方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意：AWT 并不直接调用 action 方法，AWT 直接调用的是 handleEvent 方法，handEvent 方法负责处理部件上发生的各种事件。handleEvent 行使着事件调度器的职责。当它接收到 Event.ACTION_EVENT 事件，它就会调用 action 方法。当它接收到 KEY_PRESS 事件时，它就会调用 keyDown 方法。如果被 handleEvent 调用的方法返回 false 值，handleEvent 方法将会把事件传递给父容器的 handleEvent 方法，然后由父容器的 handleEvent 方法完成分派工作。这个过程一直持续到顶层容器或者 handleEvent 方法调用的方法返回了 true 值。

在所有部件中的 action 方法的格式是：

```
public boolean action(Event event , Object whatAction)
```

在这里 event 是发生在部件上的事件，而 whatAction 指的是发生的事件。

对于按钮来说，whatAction 是被按下的按钮的标签。event 参数包含指向动作的其它信息，比如部件在何时(event.when)何地(event.target)发生动作。

警告：一定不要忘记使用 instanceof 运算符检查 event.target 变量，要确保动作是由该对象发生的。例如，如果用户希望动作是在按钮上发生的，那么用户就应确保(event.target instanceof Button)的值为 true.

既然用户已经知道了如何创建按钮和检查动作了，那么，用户就可以创建按钮小应用程序了。这里有一个非常简单的例子，该小应用程序示例使用按钮来改变背景的颜色。一种方法是将颜色的名字放在按钮标签上。然后在 action 方法中，用户寻找被按下的按钮的标签并基于标签设置小应用程序的背景颜色。例如，用户可以把背景颜色转换成兰色的按钮标签设置为 Blue(蓝色)。如果标签为 Blue 的按钮被按下时，action 方法将把背景颜色设置成蓝色。程序清单 19.1 中的小应用程序展示了具体做法。

```
import java.applet.*;
import java.awt.*;

// Example 19.1 - Button1Applet
//
// This applet creates two buttons named "Red" and "Blue". When a
// button is pressed, the background color of the applet is set to
// the color named by that button's label.
//

public class Button1Applet extends Applet
{
    public void init()
    {
        add(new Button("Red"));
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        add(new Button("Blue"));
    }

    public boolean action(Event evt, Object whatAction)
    {
// Check to make sure this is a button action. If not,
// return false to indicate that the event has not been handled.
        if (! (evt.target instanceof Button))
        {
            return false;
        }
        String buttonLabel = (String) whatAction;

        if (buttonLabel == "Red")
        {
            setBackground(Color.red);
        }
        else if (buttonLabel == "Blue")
        {
            setBackground(Color.blue);
        }
        repaint(); // Make the change visible immediately
        return true;
    }
}
```

图 19.3 显示了运行中的 Button1Applet。

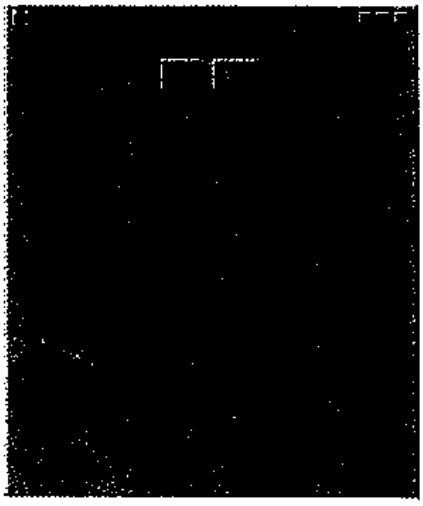


图 19.3　Button1Applet 小应用程序中改变背景颜色的按钮

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:当用户使用 JDK 1.2(就 JDK 1.1 而言不存在这个问题)编译该小应用程序时,用户将会得到一则警告信息:

```
Note: Button1Applet.java uses of overrides a deprecated API。
Recompile with "-d" for details。
1 warning
```

这个错误是因为 JavaSoft 已将该 action 方法标记为将来要从 API 删除的方法而引起的。而现在,该 action 方法仍然适用,但是不能保证将来也适用。

19.3 按照 1.1 版本的事件模式使用按钮

Java 1.0 事件模式没有提出好的面向对象设计。在 Button1Applet 小应用程序中,所有的应用程序逻辑都是小应用程序自身的。这里面没有任何部分可以作为可重用代码的编译单元。当用户想要利用其中的部分代码时,用户必须通过裁剪板来复制和粘贴其中的源代码。

虽然在 Java 1.0 中这是一个比较好的解决方法,但是 Java 1.1 的事件模式(这个模式已经带到了 Java 1.2 版本中)使得重用应用程序的代码片段更加容易。1.1 模式是依赖监听器和适配器的模式。这种模式允许用户分散程序的控制。在理想的模式中,用户的小应用程序只创建某些对象和为它们设置动作,尽量不处理这一类事情。

在 Button1Applet 小应用程序中,主要的动作是设置小应用程序的背景颜色。在 1.1 事件模式中,用户可以创建一个监听动作的对象,比如监听按钮被按下,并且通过设置小应用程序的背景颜色来响应这个动作。因为任何部件都可以改变背景颜色,所以如果仅限于用按钮来改变小应用程序的背景颜色就显得笨了些。程序清单 19.2 显示的 BGSetter 对象通过改变指定部件的背景颜色来响应动作事件。

程序清单 19.2 BGSetter.java 的源代码

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

// This class listens for an action event and then changes the
// background color of a specified component.

public class BGSetter extends Object implements ActionListener
{
    Component component;
    Color color;

    public BGSetter(Component component, Color color)
    {
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
        this.component = component;
        this.color = color;
    }

    public void actionPerformed(ActionEvent evt)
    {
        component.setBackground(color);
        component.repaint();
    }
}
```

现在所有需要小应用程序做的事情是创建一些按钮和一些 BGSetter。程序清单 19.3 显示了该小应用程序的新版本。注意 BGSetter 的实例把 ActionListener 设置成为 red 的方法。

```
import java.applet.*;
import java.awt.*;

public class Button2Applet extends Applet
{
    public void init()
    {
        Button red = new Button("Red");
        add(red);
        red.addActionListener(new BGSetter(this, Color.red));

        Button blue = new Button("Blue");
        add(blue);
        blue.addActionListener(new BGSetter(this, Color.blue));
    }
}
```

19.4　标签

标签是最简单的 AWT 部件。它们是用作声明的文本字符串。因为它们是部件，所以标签都有 action 方法，但是因为它们只做显示，所以并不产生任何动作事件。

创建标签有三种不同的办法。最简单的办法是创建空的标签：

```
public Label()
```

当然空标签没有什么大的用处，因为标签上什么也看不到。用户可以通过传递某些文本

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

给构造器来创建带文本显示的标签：

```
public Label(String labelText)
```

标签可以按照左对齐、右对齐或居中方式排列。在创建标签时可以使用 Label.LEFT、Label.RIGHT 和 Label.CENTER 变量来设置它们的对齐方式：

```
public Label(String labelText , int alignment)
```

下面的示例将标签设置为右对齐：

```
Label myLabel = new Label ("This is a right - justified label", Label.RIGHT);
```

用户可使用 setText 修改标签的文本：

```
public void setText(newLabelText)
```

用户也可以使用 getText 方法读取标签的文本：

```
public String getText()
```

用户可以使用 setAlignment 方法改变对齐方式：

```
public void setAlignment(int alignment)
throws IllegalArgumentException
```

用户也可以使用 getAlignment 方法获得标签的对齐方式：

```
public int getAlignment( )
```

图 19.4 显示的是一个标签示例。



图 19.4 标签就是文本字符串

19.5 复选框和单选按钮

除了可以用作 yes/no 或 on/off 开关外，复选框与按钮是基本一样的。每当用户单击复选框时，复选框由 on 变成 off 或者由 off 变成 on。单选按钮是与复选框最相似的部件。单选按钮也是 on/off 开关，但是它们被安排在一个互斥的组里面，在这个组中在某一时刻只能有一个按钮处于被选中状态。这就像收音机一样，同时可以有很多台，但是每次只能收听一个台。

19.5.1 创建复选框

复选框包括两个部分：标签和状态。标签是显示在框右边的文本，状态是个布尔变量，它显示该框是否被选中。在缺省情况下，复选框是处于 false 状态或 off 状态的。

复选框类有五个构造器：

```
public Checkbox( )
```

该构造器用于创建无标签的复选框。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

public Checkbox (String label)

该构造器用于创建有标签的复选框。

public Checkbox(String label,boolean state)

该构造器用于创建带有指定标签和初始状态的复选框(如果 state 为 true,则复选框的初始状态为被选中状态)。

public Checkbox(String label,boolean initialState, CheckboxGroup group)

和

public Checkbox(String label, CheckboxGroup group, boolean initialState)

这两个构造器用于创建有标签的复选框,如果 initialState 为 true,则将复选框置为被选中状态。group 参数显示该复选框归属的复选框组。CheckboxGroup 允许用户把多个复选框组织成互斥的单选按钮组。如果用户创建一个没有单选按钮的复选框,可以用 null 作为 group 的值。

19.5.2　检查和设置复选框的状态

用户可以使用 getState 方法来检查复选框是否被选中:

public boolean getState( )

例如:

```
if (myCheckbox.getState( )){
//该框已被选中
}else{
//该框未被选中
}
```

另外,用户也可以使用 setState(boolean)方法设置复选框的状态。为了使复选框被选中,用户可以使用下面的语句:

myCheckBox(true);

19.5.3　监听复选框中的变化

就像按钮一样,用户可以使用 1.0 事件模式的动作方法监视复选框的变化。但是比较好的方法是使用 ItemListener 接口。Checkbox 类有两个有关的方法用于增加和删除监听器:

addItemListener(ItemListener)
removeItemListener(ItemListener)

java.awt.Event.ItemListenr 接口有唯一的一个方法(itemStateChanged(ItemEvent))。程序清单 19.4 显示的是一个使用复选框的例子。注意内置的适配器类的使用者实现了 ItemListener 方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
/*
  *
  * CheckBoxExample
  *
  */

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class CheckboxExample extends Applet
{
    public void init()
    {

            Checkbox redBlue = new Checkbox("Red/Blue");
            add(redBlue);
            //Add an ItemListener to the red checkbox.
            //Note, the ItemListener that is added here is
            //actually an inner class.
            redBlue.addItemListener( new ItemListener(){
                    public void itemStateChanged(ItemEvent evt){
                            if (evt.getStateChange() == ItemEvent.SELECTED)
                                    setColor(Color.red);
                            else
                                    setColor(Color.blue);
                    }
            });

    }

        public void setColor(Color color){
            setBackground(color);
            repaint();
        }
}
```

在程序清单 19.4 中,加进来的监听器是一个内置类。与 Button2Applet 示例不同,在程序清单 19.3 和 19.4 中不需要两个分别的.java 文件。虽然这样限制了事件处理机制的重用,但它在用部件声明的情况下定义事件例程是非常有用的。

参见第 11 章"类"可更多地了解关于内置类的信息。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

19.5.4 创建单选按钮

单选按钮是复选框的特例。并没有单独的 RadioButton 类存在。用户可以通过把一组复选框放置在同一个复选框组中创建一套单选按钮组。CheckboxGroup 类的构造器方法不使用任何参数:

```
public CheckboxGroup()
```

当用户创建完复选框组之后,用户可以通过传递组合复选框构造器来向组中增加复选框。换句话说,如果不显式地在组中增加已有的复选框,可以创建新的隶属于组的复选框。

程序清单 19.5 创建了一个复选框组和隶属于该组的几个复选框,并且把它们全部加到小应用程序中去。

```
import java.awt.*;
import java.applet.*;

public class RadioExample extends Applet{
    public void init(){
        //create the CheckboxGroup, all the checkboxes
        //will be a member of this group
        CheckboxGroup myCheckBoxGroup = new CheckboxGroup();

        //create the checkboxes, making them members of the group
        Checkbox cb1 = new Checkbox("Favorite language is Java",
myCheckboxGroup, true)
        Checkbox cb2 = new Checkbox("Favorite language is Visual Cobol",
myCheckboxGroup, false);
        Checkbox cb3 = new Checkbox("Favorite language is Backtalk",
myCheckboxGroup, false);

        //add the checkboxes to the applet
        add(cb1);
        add(cb2);
        add(cb3);
    }
}
```

注意:当用户把复选框加进复选框组时,最后一个被加入的状态为 true 的复选框在组被显示时默认地被选中。

用户可以对每个复选框调用 getState 方法或者对 CheckboxGroup 组调用 getSelected-

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

Checkbox()。getSelectedCheckbox 方法返回当前被选中的复选框：

```
public checkbox getSelectedCheckbox()
```

> **注意**：对于使用 Java 1.0 的用户而言，用户需要用 getCurrent( )方法和 setCurrent( )方法替换 getSelectedCheckbox()方法和 setSelectedCheckbox()方法。

19.5.5 使用单选按钮

当单击复选框或单选按钮时，相应的项目事件将被调用。就复选框而言，用户必须创建一个实现 ItemListenter 接口的对象用以确定何时选中单选按钮。

当选中或释放复选框和单选项按钮时，调用 itemStateChanged 方法，它使用的参数是 ItemEvent 对象。

ItemEvent 对象可以告诉用户发生事件的对象、被选中的项目和选择的种类。getItemSelectable 方法返回发生事件的对象：

```
public ItemSeletable getItemSelectable( )
```

ItmeEvent 对象的 getItme 方法告诉用户被选中对象的值。在本示例中，它返回的是复选框或单选按钮的标签：

```
public Object getItem()
```

正像用户在程序清单 19.4 中看到那样，getStateChange 方法返回的值或者是 ItemEvent.SELECTED 或者是 ItemEvent.DESELECTED，究竟是哪个值取决于对象的选中与取消。

```
public int getStateChange( )
```

程序清单 19.6 演示了如何接收单选按钮状态改变的通知和确定哪一个项目被选中。注意在 myItemListener.itemStateChanged 方法中事件变量的用法。

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class RadioExample extends Applet{
    public void init(){
        //create the CheckboxGroup, all the checkboxes
        //will be a member of this group
        CheckboxGroup myCheckboxGroup = new CheckboxGroup();

        //create the checkboxes, making them members of
        //the group
        Checkbox cb1 = new Checkbox("Favorite language isJava", myCheckboxGroup, true);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        Checkbox cb2 = new Checkbox("Favorite language is Visual Cobol", myCheckboxGroup, false);
        Checkbox cb3 = new Checkbox("Favorite language is Backtalk", myCheckboxGroup, false);

        //add the checkboxes to the applet
        add(cb1);
        add(cb2);
        add(cb3);

        //create an ItemListner, NOTE: myItemListener is defined
        //in listing 19.6
        ItemListener listener = new myItemListener();

        //add listener to the checkboxes item list
        cb1.addItemListener(listener);
        cb2.addItemListener(listener);
        cb3.addItemListener(listener);
    }
}

class myItemListener implements ItemListener{
    public void itemStateChanged(ItemEvent event)
    {
        if (event.getStateChange() == ItemEvent.SELECTED) {
            System.out.println(event.getItem() + "has been selected.");
        } else {
            System.out.println(event.getItem() + "has been deselected.");
        }
    }
}
```

查找错误

请用户牢记：1.1 事件是包含在 jva.awt.event 包中的。这意味着用户在创建实现监听器方法的类时必须引入 java.awt 和 java.awt.event 这两个包，否则用户将得到下列错误信息：

```
RadioExample.java:31: Interface ItemListener of class myItemListener not found.
class myItemListener implements I temListener{
RodioExample.java:21:class ItemListener not found in type declaration.
ItemListener listener =new myItemListener();

2 errors
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:在 Java 1.0 事件模式中,action 方法的 whichAction 参数是一个布尔型类的实例,如果复选框被选中,该实例的值为 true;如果该复选框被取消则该实例的值为 false。

如果用户为单选按钮创建一个 action 方法,不要依靠 whichAction 参数存放该值。当单选按钮已经为 on 时单击该单选按钮,那么 whichAction 参数中包含的值应为 false,尽管该按钮仍处在 on 状态。最安全的办法是使用 getState 方法去检查单选按钮和复选框的状态。

用户也可以使用 getLabel 方法来确定哪个复选框被选中了。下面的代码片段显示了一个 action 方法,这个方法响应复选框选中的事件并取回复选框的状态。

```
public boolean action(Event evt ,Object whichAction )
  {
    if (evt.target instanceof Checkbox) //确保这是一个复选框
    {
      Checkbox currentCheckbox =(Checkbox) evt .target;
      boolean checkboxState = currnetCheckbox.getState( );
      if (currentCheckbox.getLabel( ) = = "Check me if you like Java")
      {
        if (checkboxState)
      {
      //处理"Check me if you like Java"被选中的代码
      }else {
      //处理"Check me if you like Java"未被选中的代码
      }
      return true; //事件已被处理
      }
    }
    return false ; //事件未被处理
  }
```

图 19.5 向用户显示了两个复选框和一个由三个单选按钮构成的单选按钮组。



图 19.5 复选框是方形的,选中时框内有个对号。单选按钮是圆形的,选中时圈内有个圆点

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

19.6　选择框

Choice 类提供了一个包含文本串选项的弹出式菜单。当前选项值就是当作菜单标题来显示的项。

19.6.1　创建选择框

为了创建一个弹出式选择菜单,用户必须创建一个 Choice 类的实例。因为没有选择框构造器选项,所以选择框的创建语句总是像下面的语句这样:

```
Choice myChoice = new Choice( );
```

在创建了 Choice 类后,可以用 addItem 方法为它添加字符串项目。

当用户创建了选择框后,用户可以使用 addItem 方法增加字符串项目:

```
public synchronized void addItem(String item)
throws NullPointerException
```

例如

```
myChoice.addItem("Moe");
myChoice.addItem("Larry");
myChoice.addItem("Curly");
```

用户也可以使用下面的语句从选择框列表中删除项目:

```
public synchronized void remove(int position)
public synchronized void removeAll()
```

要想修改当前被选中的项目,用户可以使用下列命令

```
public synchronized void select(int pos)
throws IllegalArgumentException

public void select(String str)
```

如果用户想让 Curly 选项被选中,用户可以使用它的名字来选中它:

```
myChoice.select("Curly");//使 "Curly"成为被选中的项目
```

用户也可以使用 Curly 在列表中的位置来选择 Curly。因为 Curly 是第三个被加进选择框的,并且选择框中的选项是从 0 开始记数的,Moe 是第 0 个,Larry 是第 1 个,所以 Curly 是第 2 个:

```
myChoice.select(2);//使第三个条目被选中
```

getSelectedIndex 方法返回被选中项目的位置:

```
public int getSelectedIndex()
```

如果 Curly 被选中,getSelectedIndex 将会返回 2。类似 getSelectedItem 方法返回被选中项目的字符串:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Public String getSelectedItem()

如果 Curly 被选中，getSelectedItem 将返回 Curly。

如果用户知道某个项目的索引值，并且希望查出与之相应的项目名称，那么就可以使用 getItem 方法来实现。

图 19.6 显示常用的选择框，图 19.7 显示的是选项菜单被拉开的选择框。



图 19.6 只显示当前选项的选择框



图 19.7 单击选择框右边的按钮可以弹出一个包含所有可能选项的菜单

19.6.2 使用选择框

像单选按钮和复选框及其它产生项目事件的部件一样，用户需要为选择框设置 ItemListener 对象来处理选择框对象的动作事件。当选择框被选中时会产生一个项目事件，即使是选择了已被选中的项目，也会产生一个项目事件。

注意：在 1.0 事件模式中，只要有选择事件发生，即使是选择了已被选中的项目，Java 都会调用选择框的 action 方法。whatAction 参数包含了被选中项目的名字。下面的代码片段给出了一个选择框的 action 方法的示例。选项是存放在小应用程序里的 String 变量中的：

```
Strign currentStooge;
public boolean action (Event event, Object whatAction)
{
//检查这是否是选择框对象，如果不是，说明事件
//尚未被处理.
if (! (event.target instanceof Choice))
{
    return false;
}
Choice whichChoice = (Choice) event.target;
//看这是否是 myChoice 的动作
if ( whichChoice = = myChoice )
{
    currentStooge = (String) whatAction;
    return true;//事件已经被处理
}
return false;//它肯定是一个不同的 Choice
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

19.7　列表框

List 类允许用户创建一个可滚动的值列表框,框中的值可一次只选一个,也可以同时选多个。用户可以随时增加和删除列表框中的项目,也可以修改当前被选中的项目。AWT 负责为用户处理滚动问题。

19.7.1　创建列表框

在创建列表框时用户可以有三种选择。List 类的缺省构造器允许用户创建禁止多项目选择的列表框:

```
public List( )
```

用户可以设置在列表框窗口中同时可见的项目的数量:

```
public List(int rows )
```

最后一种方法既可以设置显示的行数又可以决定是否允许选择多项:

```
public List(int rows, boolean multipleMode)
```

下面的代码片段创建了一个拥有 10 个可视项并可以选择多项的列表框:

```
List myList = new List(10,true);//true 意味着允许多项选择
```

当创建完列表框以后,用户就可以使用 add 方法在列表框中增加新的项目:

```
public synchronized void add (String item)
```

例如:

```
myList.add ("Moe");
myList.add("Larry");
myList.add("Curly");
```

用户也可以在列表框中指定的位置上增加项目:

```
public synchronized void add (String item , int index)
```

列表框位置是一个从 0 开始的数,所以,如果用户在位置 0 处增加项目,那么该项目将出现在列表框的顶部。如果用户试图在 -1 处或大于列表框中项目数的位置上增加项目,那么该项目将出现在列表框的末尾。下面的代码把 Shemp 项增加到列表框的顶部,把 Curly Joe 增加到末尾:

```
myList.addd("Shemp",0)          //在顶部增加项目
myList.add("Chrly Joe", -1)     //在列表框末尾增加项目
```

注意:如果用户使用的是 Java 1.0,那么就需要用 addItem()方法替换 add()方法。

19.7.2　列表框特性

List 类提供了许多不同的修改列表框内容的方法。其中 replaceItem 方法可以用新的项

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

目替换指定位置的项目：

```
myList.replaceItem("Dr. Howard",0);
//用"Dr. Howard"替换列表框中的第一个项目。
```

用户可以使用 remove 方法删除列表框中的某项：

```
public synchronized void remove (int position )
```

用户可以使用 removeAll 方法删除列表框中的所有项目：

```
public synchronized void removeAll()
```

注意：如果用户使用的是 Java 1.0，那么应该使用 delItem()方法替换 remove()方法，用 clear()方法替换 removeAll()方法。

getSelectedIndex 方法返回当前被选中项目的索引号，如果当前没有被选中的项目则返回 -1：

```
public synchronized int getSelectedIndex( )
```

用户也可以使用 getSelectedItem 方法直接获取被选中的项目：

```
public synchronized String getSelectedItem()
```

对于可以选择多项的列表框，用户可以使用 getSelectedIndexes 方法获得所有的选项：

```
public synchronized int[ ] getSelectedIndexes()
```

getSelectedItems 返回所有被选中的项目：

```
public synchronized String [ ] getSelectedItems( )
```

警告：在不允许多项选择的列表框中，用户应该只使用 getSelectedIndex 和 getSelectedItem。如果用户允许多项选择，那么用户就应该使用 getSelectedIndexes 和 getSelectedItems。

用户可以用要选择的项目的索引作为参数的 select 方法选择列表框中的任何一项。

```
public synchronized void select (int index)
```

用户可以用要释放的项目的索引作为参数的 deselect 方法释放任何项目：

```
public synchronized void deselect(int index)
```

isSelected 方法告诉用户指定索引代表的项目是否被选中：

```
public synchronized boolean isSelected(int index)
```

例如：

```
if (myList.isSelected(0)){
    //列表框中的第一项被选中
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用户可以使用 setMultipleSelections 方法打开和关闭多项选择功能:

```
public void setMultipleSelections(boolean allowMutiples)
```

如果多项选项是允许的,那么 isMultipleMode 方法返回 true:

```
public boolean isMultipleMode( )
```

例如:

```
if (myList.isMultipleMode( )){
//允许多项选择
}
```

注意:如果用户使用的是 Java 1.0,那么需要使用 allowsMultipleSelections 方法替换 isMultipleMode( )方法。

在某些时候,用户可能希望确保某个选项在列表框窗口中可见。这时可以把希望可见的项目的索引传递给 makeVisible 方法:

```
public void makeVisible(int index)
```

例如,假设列表框目前位于项目 0 处,但是用户希望项目 15 显示在窗口中,这时就可以调用:

```
myList.makeVisible(15); //使列表框中的第 15 项可见
```

19.7.3　使用列表框

当项目被选中和被释放时 List 对象都产生一个 ItemEvent 事件。ItemEvent 的 getItem 方法返回的是被选中项目的索引,而不是项目本身。当用户双击列表框中的某个项目时 List 对象产生一个动作事件。ActionEvent 的 getActionCommand 方法返回被选中项目的字符串标签。

程序清单 19.7 显示了一个完整的使用列表框的例子。注意其中的内置类 ItemListener,由它负责处理事件。

```
// This applet creates a scrolling list with several choices and
// informs you of selections and deselections using a label.
//

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class ListExample extends Applet {
        Label  listStatus;
        List scrollingList;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public void  init() {
        // First, create the List

        scrollingList = new List(3, true);
        // Now add a few items to the list
        scrollingList.add("Moe");
        scrollingList.add("Larry");
        scrollingList.add("Curly");
        scrollingList.add("Shemp");
        scrollingList.add("Curly Joe");

        // Set Shemp to be selected
        scrollingList.select(3);

        // Finally, add the list to the applet
        add(scrollingList);

        // Now create a label to show the last event that occurred
        listStatus = new Label("You selected entry Shemp");
        add(listStatus);

        scrollingList.addItemListener( new ItemListener(){
                public void itemStateChanged (ItemEvent evt){
                        String selectionString;
                        String selection;
                        int selectionNum;

if (evt.getStateChange() == ItemEvent.SELECTED){
        // selection is the index of the selected item
        selectionNum = ((Integer)evt.getItem()).intValue();
        selection = scrollingList.getItem(selectionNum);
        // use getItem to get the actual item.
        SelectionString = "You selected entry " + selection;
        // Update the label
        listStatus.setText(selectionString);
} else {
        // If this is a deselection, get the deselected item
        // selection is the index of the selected item
        selectionNum = ((Integer)evt.getItem()).intValue();
        selection = scrollingList.getItem(selectionNum);
        // use getItem to get the actual item.
        SelectionString = "You deselected entry" + selection;
        // Update the label
        listStatus.setText(selectionString);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        {
                }
            });
        }
}
```

注意:如果用户使用 1.0 事件模式,应该注意 List 类不使用 action 方法,在这一点上与前面遇到的 Java 1.0 事件模式中的用户界面部件不同。在 1.0 事件模式下 List 类使用 handleEvent 方法捕捉列表框选择和释放事件。当用户选择和释放列表框中的项目时,List 调用 handleEvent 方法。handleEvent 方法的格式是:

```
public boolean handleEvent(Event event)
```

当列表框中的某个项目被选中时,event.id 将等于 Event.LIST_SELECT,并且 event.arg 将是一个整型实例,event.arg 的值是被选中项目的索引。释放事件与选中事件是完全一致的,只是释放事件的 event.id 为 Event.LIST_DESELECT。LIST_SELECT 和 LIST_DESELECT 在 Event 类中是作为静态变量声明的,所有其它的事件类型也是这样。

在程序清单 19.8 中的小应用程序中设置了一个列表框,该列表框包含多个值,并使用标签通知用户某个项目被选中或被释放。

```
// Example 19.8 - ListApplet
//
// This applet creates a scrolling list with several choices and
// informs you of selections and deselections using a label.
//

import java.applet.*;
import java.awt.*;

public class ListApplet extends Applet
{
    Label listStatus;
    List scrollingList;

    public void init()
    {

// First, create the List

        scrollingList = new List(3, true);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// Now add a few items to the list

        scrollingList.addItem("Moe");

        scrollingList.addItem("Larry");

        scrollingList.addItem("Curly");

        scrollingList.addItem("Shemp");

        scrollingList.addItem("Curly Joe");
// Set Shemp to be selected

        scrollingList.select(3);

// Finally, add the list to the applet

        add(scrollingList);
// Now create a label to show the last event that occurred

        listStatus = new Label("You selected entry Shemp");
        add(listStatus);

    }

    public boolean handleEvent(Event evt)
    {
        String selectionString;
        Integer selection;

// Since you are handling events in the applet itself,
// you need to check to make sure the event is for the scrollingList.

        if (evt.target == scrollingList)
        {

// Check to see if this is a selection event

            if (evt.id == Event.LIST_SELECT)
            {
// selection is the index of the selected item
                selection = (Integer) evt.arg;
// use getItem to get the actual item.
                selectionString = "You selected entry " + scrollingList.getItem(
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                    selection.intValue());
// Update the label
                listStatus.setText(selectionString);
            }
            else if (evt.id == Event.LIST_DESELECT)
            {
// If this is a deselection, get the deselected item
// selection is the index of the selected item
                    selection = (Integer) evt.arg;
// use getItem to get the actual item.
                    selectionString = "You deselected entry" +
                        scrollingList.getItem(
                            selection.intValue());
// Update the label
                    listStatus.setText(selectionString);
                }
            }
            return true;
        }
    }
```

图 19.8 显示了 ListApplet 的输出。



图 19.8　ListApplet 程序允许用户选择和释放列表框中的项目

19.8　文本域和文本区

AWT 包为输入文本数据提供了两个不同的类——TextField 类和 TextArea 类。

TextField 类只能处理单行的文本，而 TextArea 类可以处理多行文本。这两个类有许多相似的方法。这两个类都是从 TextComponent 类派生出来的。

19.8.1　创建文本域

创建文本域最简单的办法就是使用空的构造器：

```
public TextField()
```

空构造器创建一个不确定列数的空文本域。如果用户想在文本域中控制多个列，那么可以使用下面的方法实现：

```
public TextField(int numColumns)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

有些时候用户希望在创建文件域的同时使用某些文本初始化文本域，这时就可以使用下述方法：

```
public TextField(String initialText)
```

还有一个方法可以将上述两种方法的功能圆满地结合在一起，创建既带有初始化文本又指定了列数的文本域：

```
public TextField(String initialText ,int numColumns)
```

19.8.2 创建文本区

创建文本区的方法与创建文本域的方法非常相似。事实上除了在指定文本区大小时既要给出文本区的列数又要给出文本区的行数以外它们是完全相同的。用户可以使用不指定行数和列数的空构造器创建一个空文本区：

```
public TextArea()
```

用户可以使用一些文本初始化文本区：

```
public TextArea(String initialText)
```

用户可以给出文本区固定的行数和列数：

```
public TextArea(int numRows, int numColumns)
```

最后，用户可以创建拥有初始化文本和固定行列数的文本区：

```
public TextArea(String initialText, int numRows ,int numColumns)
```

19.8.3 公共文本部件的特性

TextArea实际上扩展了名为 TextComponent 的类（TextField 类也扩展了该类）。TextComponent 是一个抽象类，它实现了许多有用的方法，这些方法都可以在 TextArea 类和 TextField 类中使用。

用户可能希望把文本放在某个位置上，这时用户就可以使用 setText 方法：

```
public void setText(String newText)
```

用户肯定希望获得部件中的文本，这时用户就可以使用 getText 方法来做这种事情：

```
public String getText()
```

用户可以使用 getSelectedText 方法找出被选中的文本(用鼠标使其高亮度显示的部分)：

```
public String getSelectedText()
```

用户也可以找出被选中文本的起始和结束。getSelectionStart 和 getSelectionEnd 方法可以返回被选中文本的起始或结束位置的整数：

```
public int getSelectionStart()
public int getSelectionEnd()
```

例如，如果被选中部分是从文本的头开始的，那么 getSelectionStart 应该返回 0：

```
int selectionStart,selectionEnd ;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
selectionStart = myTextField.getSelectionstart( );
selectionEnd = myTextField.getSelectionEnd( );
```

用户使用 select 方法选中文本：

```
public void select(int selectionStart, int selectionEnd)
```

如果用户希望选中所有文本，可使用 selectAll 方法：

```
public void selectAll()
```

用户也可使用 setEditable 来控制部件中的文本是否可编辑(如果不可编辑，即为只读)。

```
public void setEiditable(boolean canBeEdited)
```

如果部件是可编辑的，那么 isEdidable 方法返回 true，如果部件是不可编辑的，那么 isEditable 返回 false 。

```
public boolean isEditable()
```

19.8.4　文本域特性

文本域有一些文本区没有的特性。TextField 类允许用户设置一个响应字符，响应字符用于替代用户从键盘输入的字符。当文本域用作口令输入时，响应字符就显得很有用了；一般人此时使用通配符 * 作为响应字符。这样没有人能够看到口令的内容，能看到的只是一行通配符。用户通过调用 setEchoCharacter 可以很容易设置响应字符：

```
public void setEchoChar(char ch)
```

setEchoChar 方法最普通的用法之一是为口令显示通配符。下面的代码片段可以把响应字符设置成通配符：

```
myTextField.setEchoCharacter('*'); //在输入光标所在的屏幕位置打印 *
```

用户可以用 getEchoChar 方法找出某个文本域的响应字符：

```
public char getEchoChar( )
```

如果文本域设置了响应字符，那么 echoCharIsSet 方法就返回 true，否则就返回 false：

```
public boolean echoCharIsSet()
```

最后，用户可以使用 getColumns 方法求出文本域的列数(是可视的列数而不是有多少列文本)：

```
public int getColumns()
```

19.8.5　文本区特性

文本区也有几个自己独有的特性。文本区通常用来编辑文本，所以文本区包含一些诸如插入、追加和替换文本的方法。用户可以使用 appendText 方法在文本区的末尾增加文本：

```
public void appendText(String textToAdd)
```

用户也可以使用 insertText 方法在当前文本的任何位置插入文本。例如，如果用户在位置 0 处增加文本，那么新加入的文本就会出现在文本域的开头处。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public void insertText(String newText , int position)
```

用户也可以使用 replaceText 方法替换文本的某个部分：

```
public void replaceText(String str, int start ,int end )
```

在下面的示例中，我们使用 TextComponent 类的 getSelectionStart 方法和 getSelectionEnd 方法将 TextArea 中被选中的文本用“[CENSORED]”替换掉：

```
myTextArea.replaceText("[CENSORED]", myTextArea.getSelectionStart( ),
myTextArea.getSelectionEnd());
```

最后，用户可以用 getColumns 方法和 getRows 方法求出文本区的列数和行数：

```
public int getColumns( )
public int getRows( )
```

19.8.6 使用文本域和文本区

像 List 类一样，TextArea 类也不使用 action 方法；甚至在文本域和文本区里用户可能也不需要使用 handleEvent 方法。用户得到的 TextArea 事件是键盘事件和鼠标事件，用户可能希望 TextArea 类自己处理这些事件。用户应该创建一个按钮，当编辑完文本后就可以按下这个按钮，这时就可以使用 getText 文本获取这些被编辑过的文本。

TextField 类可以产生一个 ActionEvent 事件，也可以在用户按下 Return 按钮时使用 action 方法（这取决于用户使用的是 Java 1.1 事件模式还是使用 Java 1.0 事件模式）。同样用户可以为使用者创建一个按钮，用来发送文本输入完毕的信号（特别是当有多个文本域需要填写时尤其有用）。

程序清单 19.9 创建了两个文本域，其中一个文本域定义了响应字符，另一个文本域显示在文本域中输入的文本。

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

// TextFieldExample
// This applet creates some text fields and a text area
// to demonstrate the features of each.
//

public class TextFieldExample extends Applet
{
    protected TextField passwordField;

    protected TextArea textArea;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void init()
    {
        passwordField = new TextField(10); // 10 columns
        passwordField.setEchoChar('*'); // print '*' for input
        add(passwordField);

        textArea = new TextArea(5, 40); // 5 rows, 40 cols
        textArea.append("This is some initial text for the text area.");
        textArea.select(5, 12); // select "is some"

        add(textArea);

            passwordField.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){
            // Now, change the text in the textArea to "Your password is: "
            // followed by the password entered in the passwordField
               textArea.setText("Your password is: "+ event.getActionCommand());
                                }
                    });
    }
}
```

注意:程序清单 19.10 演示了如何在 Java 1.0 事件模式创建与程序清单 19.9 相同的两个文本域。

```
import java.awt.*;
import java.applet.*;

// TextApplet
// This applet creates some text fields and a text area
// to demonstrate the features of each.
//

public class TextApplet extends Applet
{
    protected TextField inputField;
    protected TextField passwordField;

    protected TextArea textArea;
    public void init()
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    {
        inputField = new TextField(); // unspecified size
        add(inputField);

        passwordField = new TextField(10); // 10 columns
        passwordField.setEchoCharacter('*'); // print '*' for input
        add(passwordField);

        textArea = new TextArea(5, 40); // 5 rows, 40 cols
        textArea.appendText(
            "This is some initial text for the text area.");
        textArea.select(5, 12); // select "is some"

        add(textArea);
    }

// The action method looks specifically for something entered in the
// password field and displays it in the textArea

    public boolean action(Event evt, Object whichAction)
    {
// Check to make sure this is an event for the passwordField
// if not, signal that the event hasn't been handled
        if (evt.target != passwordField)
        {
            return false; // Event not handled
        }

// Now, change the text in the textArea to "Your password is: "
// followed by the password entered in the passwordField

            textArea.setText("Your password is: " +
                event.getText());
            return true; // Event has been handled
    }
}
```

如果用户希望接收所有键盘活动的通知,可以使用另一种类型的事件监听器:TextListener。TextListener 的方法是 textValueChanged(TextEvent),因为 addTextListener() 是 TextComponent 类的方法,所以用户可以把 TextListener()方法加到 TextFields 类或 TextAreas 类中。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

19.9　滚动条

Scrollbar 类为各种滚动提供了接口。滚动条控件操纵滚动条当前的位置值。用户可以像给滚动条设置当前值一样设置滚动条的最小值和最大值。滚动控件可以用三种方式更新位置：

- 行
- 页
- 绝对位置

滚动条两头的箭头可以刷新一行内容。用户可以告诉滚动条的位置要增加多少(或减少多少)。对于刷新一行内容，缺省值是 1。

当使用鼠标单击滑块按钮与滚动箭头之间的间隙时可以刷新整页内容。用户可以告诉滚动条刷新一页时位置该加多少。

当沿某个方向拖动滑块按钮时执行绝对刷新，对于绝对刷新，用户无法控制位置值的改变，但可以控制最小值和最大值。

一个很重要的方面是 Scrollbar 类只负责刷新它自己的位置，它不能引起其它部件的滚动。如果用户希望将画布上下滚动，用户必须增加检测滚动条改变和刷新画布的代码。

19.9.1　创建滚动条

用户可以使用空构造器创建简单的垂直滚动条：

```
public Scrollbar()
```

用户也可以使用 Scrollbar.HORIZONTAL 或 Scrollbar.VERTICAL 指定滚动条的方向：

```
pubic Scrollbar(int orientation)
```

用户可以使用预定义的方向、位置、页增量、最小值和最大值创建滚动条：

```
public Scrollbar(int orientation, int position , int pageIncrement , int minimum, int maximum)
```

下面的代码创建一个最小值为 0，最大值为 100，而增量为 10 和起始位置为 50 的垂直滚动条：

```
Scrollbar myScrollbar = new Scrollbar(Scrollbar.VERTICAL,50, 10 ,0 100);
```



图 19.9　允许输入文本的文本域和文本区

19.9.2　滚动条特性

用户可以使用 setUnitIncrement 方法设置滚动条的行增量：

```
public void setUnitIncrement (int increment)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用户可以使用 getUnitIncrement( )查询当前的行增量：

```
public int getUnitIncrement()
```

用户可以使用 setBlockIncrement 方法设置页增量：

```
public void setBlockIncrement( )
```

用户可以使用 getBlockIncrement 方法查询问页增量：

```
public int getBlockIncrement( )
```

用户可以使用 getMinimum 方法和 getMaximum 方法求出滚动条滑块按钮位置的最小值和最大值：

```
public int getMinimum( )
public int getMaximum( )
```

用户可以使用 setValue 方法设置滚动条滑块按钮的当前位置：

```
public void setValue()
```

用户可以使用 getValue 方法查询滑块按钮的当前位置：

```
public int getValue ( )
```

如果滚动条是垂直滚动条，getOrientation 方法返回 Scrollbar. VERTICAL，如果滚动条是水平滚动条，getOrientation 方法的返回值为 Serollbar. HORIZONTAL：

```
public int getOrientation()
```

用户还可以使用 setValues 方法设置位置，页增量、最小值和最大值：

```
public void setValues (int position , int pageIncrement , int minimum , int maximum)
```

下面的代码将滑块按钮的位置设为 75，将页增量设为 25，将最小值设为 0，将最大值设为 500：

```
myScrollbar.setValues(75,25,0,500);
```

19.9.3 使用滚动条

Scrollbar 类产生 AdjustmentEvents 事件，并将该事件发送给 AdjustmentListener 对象。由 AdjustmentListener 接口定义的唯一一个方法是 adjustmentValueChanged：

```
public void adjustmentValueChanged(AdjustmentEvent event)
```

滚动条的调整可以有三种方式：单个单元，单元块或绝对位置(跟踪)。单个单元调整发生在用户单击滚动条两端的箭头时，单元块调整发生在用户单击箭头和滑块之间的区域时，绝对调整发生在用户拖动滑块按钮时。

AdjustmentEevent 对象的 getAdjustmentType 方法返回下列值之一：

AdjustmentEevent. UNIT _ INCREMENT, AdjustmentEvent. UNIT _ DECREMENT,
AdjustmentEevent. BLOCK _ INCREMENT, AdjustmentEvent. BLOCK _ DECREMENT 和 AdjustmentEevent. Track：

```
public int getAdjustmentType( )
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

像 List 类一样,在 Java 1.0 事件模式中 Scrollbar 类不使用 action 方法。用户必须使用 handleEvent 方法决定何时滚动条发生移动。由 Scrollbar 类产生的事件的 evt.id 的值可以是下列值之一:

- Event.SCROLL_ABSOLUTE 当滑块按钮被拖动时产生。
- Event.SCROLL_LINE_DOWN 当向上箭头或向左箭头(取决于滚动条的方向)被单击时产生
- Event.SCROLL_LINE_UP 当向下箭头或向右箭头(取决于滚动条的方向)被单击时产生
- Event.SCROLL_PAGE_DOWN 当用户单击滑块按钮和向下或向左箭头之间的区域时产生
- Event.SCROLL_PAGE_UP 当用户单击滑块按钮和向上或向右箭头之间的区域时产生。

用户可以不用关心到底这些事件中的哪一个发生。在许多情况下用户可以只知道滚动条位置的改变就可以。用户可以使用 getValue 方法求出滑块按钮的位置。

19.10　画布

Canvas 类是一个没有特殊功能的部件。它主要用于创建自定义图形部件。用户可用下面的语句创建一个 Canvas 类的实例:

```
Canvas myCanvas = new Canvas( );
```

用户可能希望创建一个可以实现用户需要的某种功能的 Canvas 子类。用户应当使用新的方法覆盖 Canvas 中的 paint 方法使得用户自己的 Canvas 能够做一些有趣的事情。

> **注意**:在缺省情况下,Canvas 没有尺寸。这样在使用部局管理器时就显得不太方便,尤其是当布局管理器中某些部件要求尺寸大小时。因此用户至少应该在 Canvas 中实现自己的尺寸方法,如果实现了 minimumSize 方法和 preferrdeSize 方法就更好了。

程序清单 19.11 创建了一个 CircleCanvas 类,该类画出一个圆并用指定的颜色填充圆。

```
import java.awt.*;

// Example 29.6 CircleCanvas class
//
// This class creates a canvas that draws a circle on itself.
// The circle color is given at creation time, and the size of
// the circle is determined by the size of the canvas.
//

public class CircleCanvas extends Canvas
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
}
    Color circleColor;

// When you create a CircleCanvas, you tell it what color to use.

    public CircleCanvas(Color drawColor)
    {
        circleColor = drawColor;
    }

    public void paint(Graphics g)
    {
        int circleDiameter, circleX, circleY;
Dimension currentSize = getSize();

// Use the smaller of the height and width of the canvas.
// This guarantees that the circle will be drawn completely.

        if (currentSize.width < currentSize.height)
        {
            circleDiameter = currentSize.width;
        }
        else
        {
            circleDiameter = currentSize.height;
        }
            g.setColor(circleColor);

// The math here on the circleX and circleY may seem strange. The x and y
// coordinates for fillOval are the upper-left coordinates of the rectangle
// that surrounds the circle. If the canvas is wider than the circle, for
// instance, we want to find out how much wider (i.e. width - diameter)
// and then, since we want equal amounts of blank area on both sides,
// we divide the amount of blank area by 2. In the case where the diameter
// equals the width, the amount of blank area is 0.

            circleX = (currentSize.width - circleDiameter) / 2;
            circleY = (currentSize.height - circleDiameter) / 2;

            g.fillOval(circleX, circleY, circleDiameter, circleDiameter);
        }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

CircleCanvas 只是一个部件,不是可执行的小应用程序。在下一章的"网格布局"部分的使用 GridBagLayout 布局管理器中示例将会用到这个新类。

19.11　部件的公共方法

Component 类定义了许多对于所有 AWT 部件和容器来说是相同的方法。几乎所有这些方法全部用于显示部件或接收输入事件。

19.11.1　部件显示方法

用户可以控制部件中许多简单的事情,比如前景色和背景色,字体是否可见。setForeground 方法和 setBackground 方法用于修改部件的前景色和背景色:

```
public void setForground(Color c)
public void setBackground(Color c)
```

虽然 setForground 方法和 setBackground 方法是为所有部件定义的,但是它们在某些 Java 实现中是不能工作的。许多 Java 实现是依赖低层窗口系统画出这些部件的,并且不能很容易地改变前景色和背景色。

用户可以用 getForground 和 getBackground 方法查询任何部件的前景色和背景色。

```
public Color getForeground( )
public Color getBackground( )
```

hide 方法和 show 方法控制部件在屏幕上是否可视。

```
public void hide( )
```

该方法只是不让部件显示出来,实际该部件仍然存在,

```
public void show( )
```

该方法使部件本身得到显示。这个方法对于框架来说是非常重要的,因为缺省情况下框架是被隐藏起来的:

```
public void setVisible(boolean showComponent)
```

> **注意**:对 Java 1.0 的使用者来说,应该使用 show( )方法替换 setVisable。

如果 showComponent 的值为 true,则显示部件。如果 showComponent 的值为 false,那么部件就会被隐藏起来。

setFont 方法用于改变部件的字体。这个方法只适用于显示文本的部件:

```
public void setFont(Font f)
```

用户可以使用 getFont 方法查询部件当前使用的字体:

```
public Font getFont( )
```

Component 类还允许用户访问字体的度量:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public FontMetrics getFontMetrics(Font font)
```

19.11.2 部件的位置和尺寸设置

部件的尺寸和位置是受布局管理器支配的。部件可以返回它的首选尺寸和最小尺寸,但是布局管理器最终决定部件的实际尺寸。另外布局管理器还决定部件的位置(它的 x 和 y 坐标)。当布局管理器决定了部件的位置和尺寸以后,由布局管理器调用部件内的方法重新设置部件的尺寸和位置。

getMinimumSize 方法返回部件必须给出的最小宽度和高度,preferredSize 返回首选宽度和高度:

```
public Dimension getMinimumSize( )
public Dimension preferredSize( )
```

getSize 方法返回部件的实际宽度和高度:

```
public Dimension getSize( )
```

setLocaion 方法设置部件显示区左上角的 x 和 y 坐标:

```
public void setLocation(int x,int y)
```

这些坐标是相对于父部件的空间的。例如,如果某个部件被移动到 0 ,0 并且它的父部件位于屏幕的 100,150,那么该部件将被实际地画在 100,150 处。图 19.10 阐述了部件的坐标,父部件的坐标和实际的屏幕坐标之间的关系。

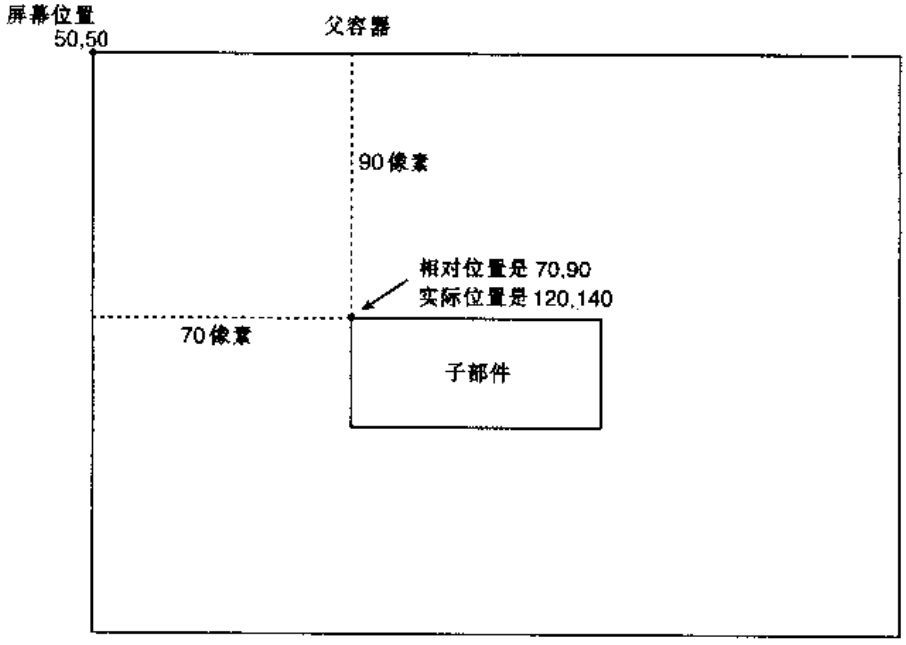


图 19.10 部件的坐标是相对于它的父容器的

如果用户希望查询部件相对它的父容器的显示区位置,可以使用 location 方法:

```
public Point location( )
```

getComponentAt 方法寻找包含 x,y 点的部件:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public Component getComponentAt(int x,int y)
```

如果该点不在这个部件内部,那么 locate 方法返回 null。如果该点在这个部件内部并且部件包含子部件,那么它就像是一个包含该点的子部件。如果找到一个部件,locate 方法返回那个部件,如果没有找到部件,它就返回当前部件。注意 locate 方法只查找一级深度的子部件。当用户获得一个子部件之后,用户可以重复查找。

下列的方法查找屏幕上占用特定的 x,y 坐标的部件。如果 locate 返回一个容器,那么它将继续查找容器内的部件直到找到要找的部件为止。

```
public Component findComponent(int x, int y)
{
    //找出该 x,y 点落在哪个部件中
    Component whichComp = getComponentAt(x , y);
    //如果部件是一个容器,那么继续查找哪个部件包含该
    //x,y 点
    while( whichComp instanceof Container){
        //如果用户必须在容器内查找,那么必须将该 x , y
        //点调整为相对于该容器的坐标
        x - = whichComp.getLocation( ).x;
        y - = whichComp.getLocation( ). y;
    Component nextComp = whichComp.getComponentAt(x , y);

        //如果 locate 返回部件本身,那么查找宣告结束
        if (nextComp = = whichComp) break;
            whichComp = nextComp;
        }
    return whichComp;
}
```

注意:如果用户使用的是 Java 1.0,上面使用到的某些方法需要用下面表中相应的方法替换:

Java 1.1&1.2	Java 1.0
getSize	size
getMinimumSize	minimumSize
getComponentAt	locate
setLocation	move

这种替换将产生下列方法:

```
public Component findComponent(int x, int y)
{
    //找出该 x,y 点在哪个部件中
    Component whichComp = locate(x , y);
    //如果部件是一个容器,那么继续查找哪个部件包含该 x,y 点
    while( whichComp instanceof Container){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            //如果用户必须在容器内查找,那么必须将该 x , y
            //点调整为相对于该容器的坐标
            x - = whichComp.location( ).x;
            y - = whichComp. location ( ). y;
            Component nextComp = whichComp. locate (x , y);
            //如果 locate 返回部件本身,那么查找结束

        if (nextComp = = whichComp) break;
                whichComp = nextComp;
        }
        return whichComp;
    }
```

19.11.3 部件布局和送显的方法

用户可能熟悉了部件送显(在屏幕上画出)的主要方法。它们是 repaint , update 和 paint。

public void repaint( )

该方法要求图像尽快地重画。最终是要调用 update 方法,但不是马上调用。

public void repaint(int x, int y ,int width, int height)

部件重画的部分只是由参数指定的区域。

public void repaint (long tm)

该方法要求部件在 tm 毫微秒内被重画。

public void repaint(long tm,int x, int y, int width, int height)

该方法要求在 tm 毫微秒内重画部件的指定部分。

public void update(Graphics g)

该方法将部件重画引进图形设备描述体 g 中。缺省的 update 方法删去图形设备描述体并调用 paint 方法。

public void paint(Graphics g)

该方法在图形设备描述体 g 上重画该部件。

当部件是由布局管理器布局时,它们被标记为有效的。即它们都被检查过和布局过。如果部件改变了尺寸或由于其它方面的原因需要改变当前的布局,那么就可以通过 invalidate 方法把该部件标记为无效的:

public void invalidate( )

使部件无效就是将部件标记为被修改了,当下一次调用部件的(或父部件的)validate 方法时,将会再一次执行部件布局。validate 方法的格式是:

public void validate( )

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

validate 方法也使用该部件的所有子部件的 doLayout 方法:

```
public void doLayout()
```

部件缺省的 layout 方法不做任何事情。在某个容器内,layout 方法引起布局管理器重新计算该容器包含的每个部件的位置。

用户可以通过 getParent 方法获得对部件的父容器的引用:

```
public Container getParent()
```

用户可以引用小应用程序的父框架,为此,需要反向追查小应用程序的父容器直至找到某个框架。用户可能得到一个无法预言的结果,有时可能找不到。下面的循环试图找出小应用程序的父框架:

```
Container parent = getParent( );
//向上追踪父框架直到没有父框架或者找到了某个框架为止
while ((parent! = null) && ! (parent instanceof Frame))
{
parent = parent.getParent( );
}
//此时 parent 或者为空或者为小应用程序的父框架
```

19.11.4　部件输入事件

部件可以接受大量的动作,所以部件都有许多增加事件监听器的方法。每个部件都有下述方法:

```
public void addComponentListener(ComponentListener)
public void addFocusListener(FocusListener)
public void addKeyListener(KeyListener)
public void addMouseListener(MouseListener)
public void addMouseMotionListener(MouseMotionListener)
```

正像用户已经猜到的那样,FocusListeners 方法用于监听焦点的变化(获得或失去),KeyListeners 监听键盘输入,MouseListeners 监听鼠标进入、离开或在某个区域点击,MouseMotionListeners 监听鼠标的移动。但是 ComponentListener 是干什么的呢?

ComponentListeners 用于监听部件位置和尺寸的变化以及显示与隐藏的切换。使用所有这些监听器,用户可以监视任何部件的任何动作。

注意:在 1.0 事件模式中,handleEvent 方法用于通知部件接收输入,handleEvent 方法实际是较长的事件处理方法链中的一个部分。

```
public void deliverEvent(Event evt)
```

该方法传送一个事件给这个部件。这是初始的指向事件处理链中的某个事件的项。

```
public void postEvent(Eventevt)
```

该方法把事件传递给 handleEvent 方法。如果 handleEvent 方法返回 false,那么这个方法用父部件的 postEvent 方法把事件传递给父部件。如果 postEvent 返回 true,说明该事件被成功地处理了。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

public boolean handleEvent(Event evt)

该方法检查该事件并基于事件类型调用下列方法之一：mouseEnter，mouseExit、mouseMove、mouseDrag、mouseDown、mouseUp、keyDown、keyUp、action、gotFocus 和 lostFocus。

用户可以使用 disable 方法禁止部件接收输入事件：

public void disable( )

要想让部件重新具有接收输入事件的能力，用户可以调用 enable 方法：

public void enable( )

如果部件处于有效状态，那么 isEnabled 方法返回 true

public boolean isEnabled( )

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 20 章　深入研究异常和事件

本章内容：

20.1　Java 的异常
20.2　Java 的 Error 类
20.3　Java 的事件
20.4　事件处理技术
20.5　键盘和鼠标事件
20.6　Java 1.0 的事件模式

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

20.1 Java 的异常

当用户使用 Java 编写小应用程序或应用程序时，或迟或早都会遇到异常。异常是一个特殊的对象类型，当程序出现错误时系统就会创建这种类型的对象。当 Java 创建了 exception 对象以后，Java 将它发送给用户的应用程序，产生异常的动作在英文中叫做“throwing”，当异常产生后由程序去捕捉这个异常。捕捉异常的工作由被称作“异常处理”的代码负责。在本章中我们将介绍关于错误处理对象的一些重要的内部信息。

Java 的另一个重要的活动类型是事件。事件代表在运行程序期间用户执行的动作，比如单击按钮或移动鼠标。用户马上可以看到，Java 有几种处理事件的方法。本章将深入地讲解 Event 类和如何在工程中使用 Event 类。

在前面的某些章中，我们曾经提到过异常和如何在程序中处理异常。异常是由方法调用不能完全成功产生的(比如试图让某个线程休眠)。在这种情况下，Java 产生一个名为 InterruptedException 的异常(见程序清单 20.1)。

```
try
{
    Thread.sleep(500);
}
catch (InterruptedException e)
{
    String err = e.toString();
    System.out.println(err);
}
```

正如用户在程序清单 20.1 中看到的那样，用户可以把可能产生异常的代码放在 try 块中并把异常处理代码放在 catch 程序块中。在程序清单 20.1 中 try 块的第一行试图让当前线程休眠 500 毫秒。如果该线的 sleep 运行不成功，那么 sleep( )方法就会产生一个 InterruptedException 异常。当异常产生时，Java 忽略 try 块中剩下的代码直接跳到 catch 块中进行异常处理。另一方面，如果 sleep 运行成功，Java 将执行 try 块中的全部内容并跳过 catch 块。

> **注意**：catch 块不是在继续执行程序，它实际上是在处理由 Java 产生的异常。在程序清单 20.1 中用户可以看到异常对象是在 catch 关键字后面的圆括号中被捕获的。这很像参数被传递给方法。在 catch 块中，“参数”的类型是 InterruptedException，“参数”名是 e。如果有必要的话，用户可以通过 e 对象访问异常对象的方法。在本例中程序调用了 e 对象的 toString( )方法以获取代表异常对象的字符串。

Java 定义了许多可以由 Java 类中的方法产生的异常。那么用户怎么才能知道哪个异常是需要处理的？如果用户调用一个明显地说明可能会产生异常的方法，这时 Java 将坚持用户

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

提供这种或那种异常处理手段。如果用户没有按照这个要求编写程序，那么用户编写的类将无法通过编译。用户将会收到一条错误信息，该信息指示用户的程序可能会产生异常(见图 20.1)。

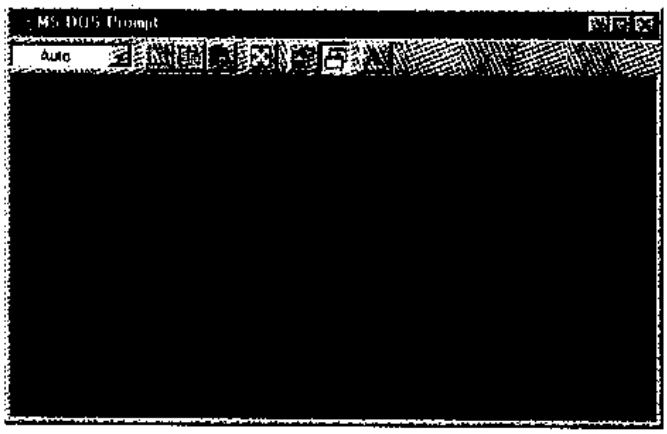


图 20.1　如果用户在小应用程序中没有编写异常捕获和处理代码，Java 编译器将产生一则错误信息

虽然编译器的错误信息是查找错误的线索，但是聪明的程序员一般在使用方法之前将会在 Java 文档中查看要使用的方法。这样程序员在实际使用该方法之前就已经知道是否有必要为它编写异常处理代码。如果用户有兴趣查看某个包中定义的异常，可以将该包的 Java 在线文档中的类和异常全部被列出来(如图 20.2 所示)。

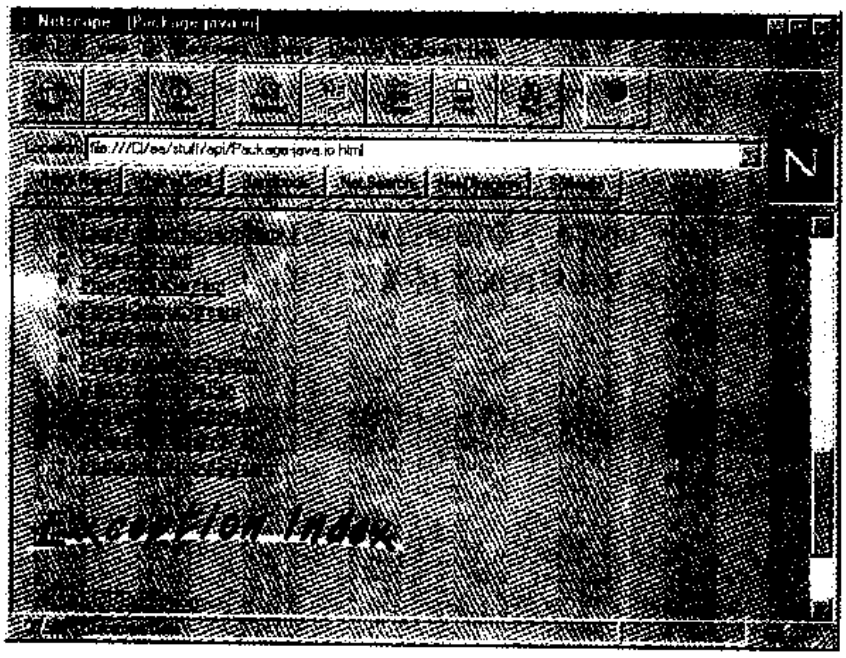


图 20.2　Java 的在线文档列出了类中的方法可能产生的异常对象

提示：在 Sun 公司网页上的在线文档是经常被更新的。要想跟上 Java 发展的步伐，可以在浏览器中为 http://www.javasoft.com/products/JDK 网址设置一个书签以便于时常到该网站上访问一下。

在线文档还列出了构成某个包的所有方法。通过查看文档中的方法(如图 20.3 所示)，用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

户可以看到方法使用的参数类型,返回值的类型和方法是否产生异常。如果方法可以产生异常,那么用户的代码必须正确地处理这种类型的异常,否则代码将无法通过编译。

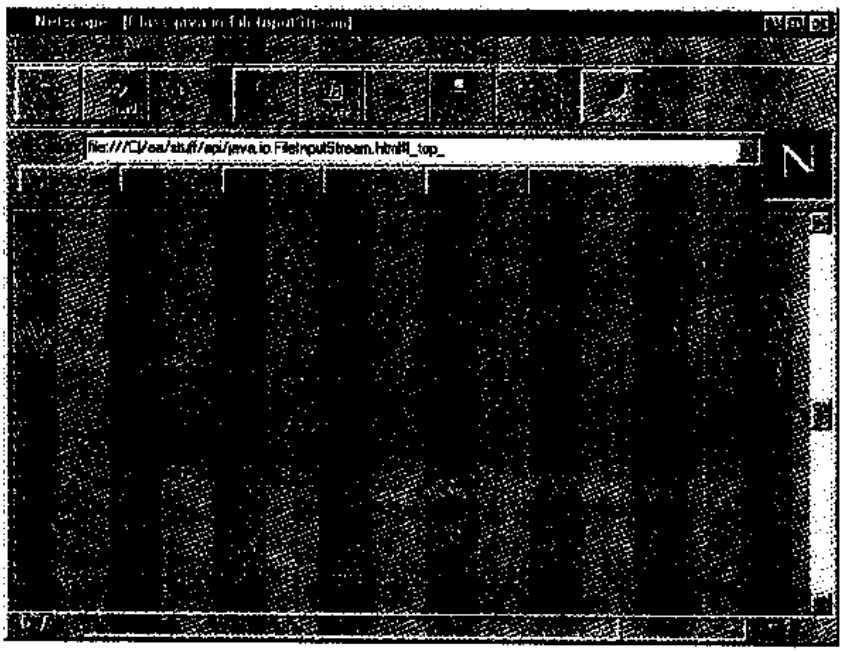


图 20.3 方法的在线文档显示了方法可能产生的异常

20.1.1 抛出异常

一种方便的异常处理办法是不必在产生异常的方法中处理异常。例如,当程序清单 20.1 试图将线程设置成休眠状态,如果设置失败,该方法就产生一个由程序在 catch 块中处理的异常。

但是出于某种原因,如果用户不希望捕获像在 read 方法中那样的异常该怎么办? 如果用户不知道该怎样处理异常,用户就可能不希望捕捉它。在这种情况下,Java 允许用户简单地传递这种异常的产生,也就是说向方法层次的上层抛出异常。程序清单 20.2 显示了一种使用 IOException 异常的处理办法。

```
protected void MyMethod(){
    try
    {
        DoRead();
    }
    catch (IOException e)
    {
        String err = e.toString();
        System.out.println(err);
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
protected void DoRead() throws IOException
{
    System.in.read(buffer, 0, 255);
}
```

read( )方法抛出一个 IOException 异常，所以用户可以在 try/catch 块中调用它。在程序清单 20.2 中，对 read( )的调用被移动到了名为 DoRead( )方法中，其中 DoRead( )方法并不直接处理 IOException 异常。DoRead( )方法说明它可以产生一个 IOException 并将该异常传回相应的调用方法。Java 知道 DoRead( )想要传递异常，因为 DoRead( )为其签名增加了短语 throws IOException。抛出异常并不会最终地免除用户对异常的处理。注意在程序清单 20.2 中，异常仍然会在 myMethod( )方法中得到处理。

简而言之，用户可以通过两种途径处理异常：

- 在调用可能产生异常的功能中直接编写 try 和 catch 程序块。
- 声明抛出异常的方法，在这种情况下，用户需要在调用“抛出”的方法中编写 try 和 catch 程序块，如程序清单 20.2。

20.1.2　联合方案

在处理异常时，人们经常既使用在代码中处理异常的办法，也使用向函数传递异常的处理办法。Java 允许联合使用这两种处理办法，这样做可以使程序的不同部分用恰当的方法处理异常。要想使用这种联合的异常处理方案，需要在程序中包含一个 try/catch 程序块和一个在某个中方法中声明的 throws 短语。程序清单 20.3 显示了这种技术的示例。

```
protected void MyMethod() throws IOException{
    try
    {
        DoRead();
    }
    catch (IOException e)
    {
        String err = e.toString();
        System.out.println(err);
          throw e;
    }
}
```

正如用户在前面几个例子中看到的那样，exception 对象可以在程序中做大量的移动。它们可以从一个方法中跳到另一个方法，可以跨越方法的层次直到找到最终处理它们的方法。如果异常跳到了 Java 系统，那么系统就使用缺省的方式处理它，通常是产生错误信息。当在浏览器中运行小应用程序时，用户可能没有机会看到这些错误信息。更糟糕的是，某些兼容

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java 的浏览器在处理异常时有些差异。某个浏览器可能忽略该异常并继续运行，而有的浏览器可能因此而卡住。最好的方案应当是把在程序中可能发生的异常通盘地加以考虑。这样就可以保证浏览器不会受这些错误的影响。

20.1.3 异常的类型

Java 定义了许多异常对象，当用户调用能够抛出异常的函数时，其中一些异常在用户的程序里必须处理。而其它的一些异常是由系统产生的，比如内存分配失败、表达式试图被零除、空值的不正确使用等等。用户可以有选择地监视第二种异常或者让 Java 虚拟机处理它们。所有这些异常类型都是从 RuntimeException 类派生出来的。

当异常出现在程序中之前用户就应该率先找出在程序中可能出现异常的地方。这些地方通常与用户的输入有关，这是让人最难预料的。程序员应该知道异常抛出是由于程序错误导致的。某些常见的异常在用户的小应用程序中需要监视，表 20.1 列出了这些异常。

表 20.1 常见的异常

异　　常	引 发 原 因
ArithmeticException	数学错误，比如被零除
ArrayIndexOutOfBoundsException	错误的数组索引
ArrayStoreException	程序试图在数组中存放错误的数据类型
FileNotFoundException	试图访问不存在的文件
IOException	普通的 I/O 错误，比如无法从文件中读取信息
NullPointerException	引用空对象
NumberFormatException	失败的字符串和数字转换
SecurityException	小应用程序试图执行浏览器安全设置不允许的动作
OutOfMemoryException	内存不足，不能为新对象分配空间
StackOverflowException	系统用尽了堆栈空间
StringIndexOutOfBoundsException	程序试图访问字符串中不存在的字符

提示：用户可以通过设置各种类型的 Exception 的 catch 块来捕捉所有的异常，比如：

```
catch(Exception e)
```

用户可以使用异常的 getMessage( )方法(从 Throwable 超类继承来的)获取与用户截获的异常有关的信息。

所有 Java 的异常都是从 Throwable 类扩展出来的类，所有的异常和错误对象都可以跟踪它们的祖先。(用户将在本章后面的"Java 的 error 类"部分了解 error 对象)Throwable 类定义了三个很有用的方法，用户可以通过调用这三个方法来获取关于异常的信息：

- getMessage( )　　可以得到一个详细描述异常的字符串
- toString( )　　把对象转换为可以在屏幕上显示的字符

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- printStackTrace( )　　显示引起异常的方法调用层次

程序清单 20.4 显示了一个调用这三种方法的 catch 块，图 20.4 显示了这个 catch 块产生的输出。（注意在 NumberFormatException 中，getMessage( )方法返回一个空字符串。）

程序清单 20.4　EXT20_04.TXT—调用 throwable 对象的方法

```
catch (NumberFormatException e)
{
    System.out.println();
    System.out.println("Here's getMessage()'s string:");
    System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
String str = e.getMessage();
    System.out.println(str);
    System.out.println();
    System.out.println("Here's toString()'s string:");
    System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
    str = e.toString();
    System.out.println(str);
    System.out.println();
    System.out.println("Here's the stack trace:");
    System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
    e.printStackTrace();
}
```

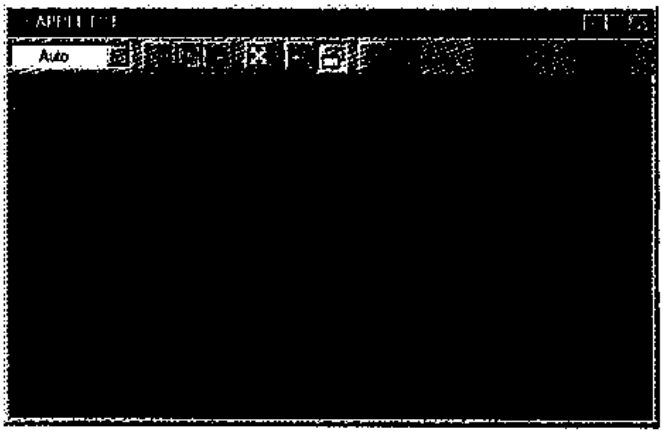


图 20.4　由程序清单 20.4 中的 catch 块产生的输出

异常可以分为三种主要的类型：

- 从 Exception 类直接派生出来的异常类
- 运行时刻类
- I/O 异常类

按照在类层次中的顺序我们把一些异常类列举如下。类名后面的括号中显示的是定义该异常类的包。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Throwable (java.lang)
Exception (java.lang)
AWTException (java.awt)
NoSuchMethodException (java.lang)
InterruptedException (java.lang)
InstantiationException (java.lang)
ClassNotFoundException (java.lang)
CloneNotSupportedException (java.lang)
IllegalAccessExcption (java.lang)
IOException (java.io)
EOFException (java.io)
FileNotFoundException (java.io)
InterruptedIOException (java.io)
UIFDataFormatException (java.io)
MalformedURLException (java.net)
ProtocolException (java.net)
SocketException (java.net)
UnknownHostException (java.net)
UnknownServiceException (java.net)
RuntimeException (java.lang)
ArithmeticException (java.lang)
ArrayStoreException (java.lang)
ClassCastException (java.lang)
IllegalArgumentException (java.lang)
IllegalThreadStateException (java.lang)
NumberFormatException (java.lang)
IllegalMonitorStateException (java.lang)
IndexOutOfBoundsException (java.lang)
ArrayIndexOutOfBoundsException (java.lang)
StringIndexOutOfBoundsException (java.lang)
NegativeArraySizeException (java.lang)
NullPointerException (java.lang)
SecurityException (java.lang)
EmptyStackException (java.util)
NoSuchElementException(java.util)

注意:该异常列表中显示的只是由原始的 Java 类创建的异常。最新版本的 Java 增加了许多新类,所以也增加了许多新的异常。

20.1.4 确定要处理的异常

有经验的程序员通常都知道他们的代码何时产生某种异常。当用户第一次编写带有异常处理代码的小应用程序,用户可能不能保证异常被监视到,找出这种信息的一种方式是测试小

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

应用程序，看究竟产生什么异常。

程序清单 20.5 是一个名为 ExceptionApplet 的小应用程序，它接收用户输入的两个整数，然后把它们相除，最后显示整数结果（余数舍掉）。因为该小应用程序必须处理输入，所以出现问题的可能性很高。而 ExceptionApplet 没有包含异常处理代码。

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class ExceptionApplet extends Applet
{
    TextField textField1, textField2;
    String answerStr;
    public void init()
    {
        textField1 = new TextField(15);
        add(textField1);
        textField2 = new TextField(15);
        add(textField2);
        answerStr = "Undefined";

                //create an inner ActionListener class
                ActionListener listener = new ActionListener(){
                    public void actionPerformed(ActionEvent evt){
                        performAction();
                    }
                };

                textField1.addActionListener(listener);
                textField2.addActionListener(listener);
    }

    public void paint(Graphics g)
    {
        Font font = new Font("TimesRoman", Font.PLAIN, 24);
        g.setFont(font);
        g.drawString("The answer is:", 50, 100);
        g.drawString(answerStr, 70, 130);
    }

    public void performAction()
    {
        String str1 = textField1.getText();
        String str2 = textField2.getText();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            int int1 = Integer.parseInt(str1);
            int int2 = Integer.parseInt(str2);
            int answer = int1 / int2;
            answerStr = String.valueOf(answer);
            repaint();
        }
    }
```

注意:如果用户使用的是 1.0 事件模式,那么就需要使用下面的 perfomAction 方法替换上面的嵌入类:

```
public boolean action(Event evt ,Object arg)
    {
        String str1 = textField1.getText( );
        String str2 = textField2.getText( );
        int int1 = Integer.parseInt(str1);
        int int2 = Integer.parseInt(str2);
        int answer = int1/int2;
        answerStr = String.valueOf(answer);
        repaint( );
        return true;
    }
```

用户可以把这个小应用程序作为开发健壮的小应用程序的起点。当用户使用小应用程序浏览器运行这个小应用程序时,用户将会看到如图 20.5 所示的窗口。在每个文本框中输入一个数字后按下回车键。这时程序用第二个数去除第一个数并把结果显示出来(如图 20.6 所示)。

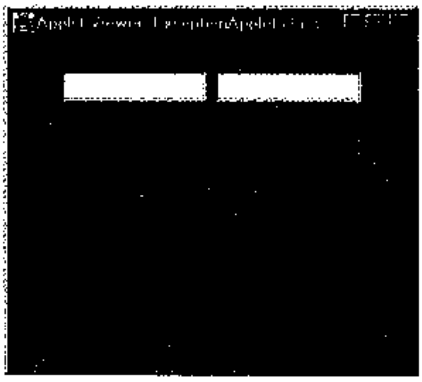


图 20.5 在小应用程序浏览器下运行的 ExceptionApplet

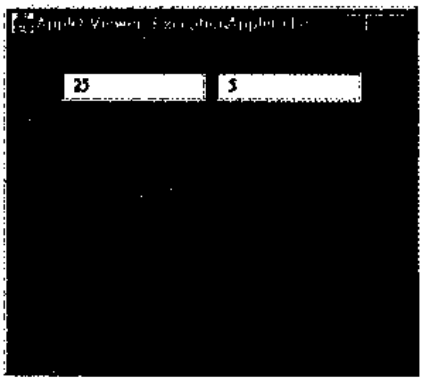


图 20.6 ExceptionApplet 用第二个数除第一个数

只要用户在文本框中输入有效的数字,程序就会正常地运行。

但是如果用户在一个或两个文本框为空时按下了回车键将会发生什么事情呢? 当 action( )方法试图将空文本框中的内容转换成整数值时,Java 立刻会抛出 NumberFormatException 异常。用户可以在运行小应用程序浏览器命令行窗口中看到这个异常,如图 20.7 所示。正如

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用户可以从图中看到的那样，Java 显示了几行跟踪异常的信息。其中第一行（以 Exception 开头的那行）告诉用户现在遇到的异常类型。

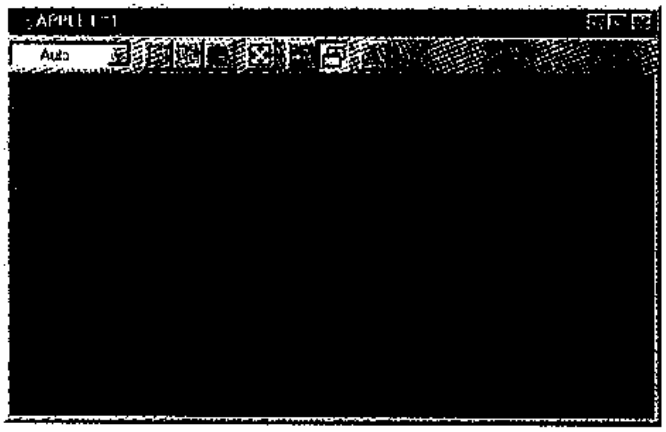


图 20.7　Java 报告了一个 NumberFormatException 异常

注意：现在用户应当知道：不必捕捉 Java 产生的每一个异常。当用户没有提供处理不要求捕捉的异常的代码，Java 会在内部捕捉该异常。当这种情况发生在小应用程序浏览器下运行小应用程序时，用户将在命令行窗口中看到异常错误出现。如果运行在网页浏览器中的小应用程序产生异常时，用户可能根本就觉察不到它，因为小应用程序一般不会崩溃或显示错误信息，这只是因为它不执行显示这个异常产生的命令。

20.1.5　捕捉运行时刻异常

用户现在应该知道，如果不在文本框中输入内容或输入无效内容（比如字符串 one）将会引发 NumberFormatException 异常。为了保证小应用程序不出现意外，用户现在需要编写处理异常的代码。下面是增添异常处理代码的步骤：

1. 将 ExceptionApplet 加载到用户的文本编辑器中。
2. 用程序清单 20.6 中版本替换原来的 action 方法。

```
public boolean action(Event evt, Object arg)
{
    String str1 = textField1.getText();
    String str2 = textField2.getText();
    try
    {
        int int1 = Integer.parseInt(str1);
        int int2 = Integer.parseInt(str2);
        int answer = int1 / int2;
        answerStr = String.valueOf(answer);
    }
    catch (NumberFormatException e)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
            answerStr = "Bad number!";
        }
        repaint();
        return true;
    }
```

3. 在类声明行中将类名修改成 ExceptionApplet2。
4. 将新的小应用程序存储在名为 ExceptionApplet2.java 的文件中。
5. 加载 EXCEPTIONAPPLET.HTML 文件。
6. 将文件中的所有的 ExceptionApplet 修改成 ExceptionApplet2。
7. 将该文件另存为 EXCEPTIONAPPLET2.HTML 文件。

在程序清单 20.6 中,现在的 action( )方法使用 try 和 catch 程序块优雅地处理着 NumberFormatException 异常。图 20.8 显示了现在如果空着文本框将会发生的情况。当该程序第一次调用 String.valueOf( )方法时,Java 产生 NumberFormatException 异常,该异常引导程序跳到 catch 块。在 catch 块中程序将显示字符串设置为 Bad number! 而 repaint( )调用保证这则信息显示在屏幕上。

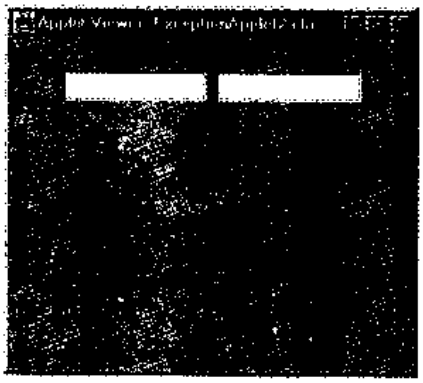


图 20.8 ExceptionApplet2 优雅地处理 NumberFormatException 异常

20.1.6 处理异常

现在,用户可以输入数字到 ExceptionApplet2 的文本框,然后得到结果。也许用户可能在第二个文本框中输入了零,Java 试图用零除第一个数,这时用户就会得到一个 ArithmetricException 异常。这是怎么回事,不是已经使用 catch 处理 NumbeFormatException 异常了吗? 现在又有另外一个异常需要处理了。

一个好消息是用户不必局限在一个 catch 块中。事实上用户可以为每一个用户认为可能产生的异常都创建一个 catch 块。为了让用户看清楚如何在程序中使用多个 catch 块,我们一步步地介绍:

1. 把 ExceptionApplet 加载到文本编辑器中。
2. 用程序清单 20.7 所示的代码替换 action( )方法。

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
public boolean action(Event evt, Object arg)
{
    String str1 = textField1.getText();
    String str2 = textField2.getText();
    try
    {
        int int1 = Integer.parseInt(str1);
        int int2 = Integer.parseInt(str2);
        int answer = int1 / int2;
        answerStr = String.valueOf(answer);
    }
    catch (NumberFormatException e)
    {
        answerStr = "Bad number!";
    }
    catch (ArithmeticException e)
    {
        answerStr = "Division by 0!";
    }
    repaint();
    return true;
}
```

3. 在类声明行中，将类名修改成 ExceptionApplet3。

4. 将新的小应用程序另存为 ExceptionApplet3.java。

5. 加载 EXCEPTIONAPPLET3.HTML 文件。

6. 将文件中的所有 ExceptionApplet 全部改成 ExceptionApplet3。

将文件另存为 EXCEPTIONAPPLET3.HTML。

如果用户仔细查看程序清单 20.7 可以发现：原来的 action( )方法现在被定义成了两个 catch 程序块，其中一个用于处理 NumberFormatException 异常，另一个用于处理 ArithmeticException 异常。使用这种方式，程序可以使用一个 try 块监视两个潜在的问题。图 20.9 显示了当用户试图让某数被零除的异常处理结果。如果用户发现程序中还有另外的异常，用户可以增加另外的 catch 块。

确保首先捕捉到某个特定的异常很重要。例如，如果用户希望捕捉到 FileNotFoundException 异常，用户首先应该在捕捉到其它更普通的 IOException 异常之前处理它们。

注意：虽然异常处理是一个防止程序崩溃的强有力的工具，但是用户应该只在无法控制异常的产生时使用它们，比如在处理用户输入时使用异常处理。如果是因为程序错误而引起小应用程序出现异常，那么用户应该扎扎实实地解决这个问题，而不应该使用异常处理来解决这个问题。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

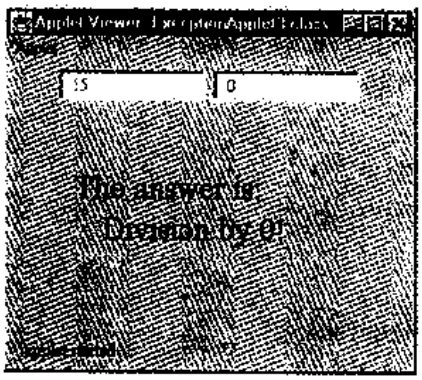


图 20.9 ExceptionApplet3 捕捉被零除错误

提示:有时用户可能希望某个特殊的代码块不管是否出现异常都能够被执行。这种做法是可行的,用户可以在最后一个 catch 块之后增加 finally 程序块。finally 块可以在 try 块或 catch 块结束后得到执行,如程序清单 20.8 所示。

```
try
{
    // The code that may generate an exception goes here.
}
catch (Exception e)
{
    // The code that handles the exception goes here.
}
finally
{
    // The code here is executed after the try or
    // catch blocks finish executing.
}
```

20.1.7 创建用户自己的异常类

虽然 Java 提供了处理用户在一般情况下可以想像得到的各种错误的异常类,但是 Java 语言的设计者。不可能知道用户要编写什么类型的代码,也不可能知道用户的代码可能出现什么样的错误。例中用户可以编写一个统计某个范围内的两个数。如果输入的数超出了选定的范围,则程序可能会抛出一个自定义的异常,比如说叫 NumberRangeException。

为了创建和抛出用户自己定义的异常,用户必须首先为异常定义一个类。通常,用户可以扩展 Java 的 Exception 类。程序清单 20.9 显示了如何定义前面提到过的 NumberRangeException 类。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public class NumberRangeException extends Exception
{
    public NumberRangeException(String msg)
    {
super(msg);
    }
}
```

正如用户看到的那样,定义新的异常只需要做很少的工作。事实上用户只要为类创建一个构造器就行。注意 NumberRangeException 类的构造器接受的是字符串型的参数。该字符串是调用类的 getMessage( )方法(该方法是由该类通过 Exception 类从 Throwable 类继承来的)时返回的详细信息。在构造器中,该字符串被传递给 NumberRangeException 的超类(Exception),Exception 类将该字符串再传递给 Throwable 类,在这里它是被作为类的数据成员存储的。现在在用户的程序里,一旦自定义异常的条件出现,用户就可以创建并抛出用户的异常类的对象。

程序清单 20.10 是一个测试新的 NumberRangeException 的小应用程序。当用户运行这个小应用程序时,可试着在每个文本框中填写一个数。如果用户按照提示输入两个范围在 10～20之间的数,则小应用程序将合计这两个数并在屏幕上显示该结果。否则,小应用程序将产生 NumberRangeException 异常并显示错误信息,如图 20.10 所示。

提示:当用户编译 ExceptionApplet4 小应用程序时,应确保 NumberRangeException.java 文件与该小应用程序的源代码在同一个目录下。否则 Java 编译器可能找不到它。用户也可以在 CLASSPATH 环境变量增加小应用程序的路径。

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class ExceptionApplet4 extends Applet
{
    TextField textField1, textField2;
    String answerStr;
    public void init()
    {
        textField1 = new TextField(15);
        add(textField1);
        textField2 = new TextField(15);
        add(textField2);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        //create an ActionListener. This will be registered with
        //the TextFields and when 'nter' is pressed the actionPerformed()
        //method will be called.
        ActionListener listener = new ActionListener(){
                public void actionPerformed(ActionEvent evt)
                {
                        try
                        {
                                int answer = CalcAnswer();
                                answerStr = String.valueOf(answer);
                        }
                        catch (NumberRangeException e)
                        {
                                answerStr = e.getMessage();
                        }
                        repaint();
                        return true;
                }
        };

        //register the listener with the TextFields
        textField1.addActionListener(listener);
        textField2.addActionListener(listener);

        answerStr = "Undefined";
        resize(500, 200);
    }

    public void paint(Graphics g)
    {
        Font font = new Font("TimesRoman", Font.PLAIN, 24);
        g.setFont(font);
        g.drawString("Enter numbers between", 40, 70);
        g.drawString("10 and 20.", 70, 90);
        g.drawString("The answer is:", 40, 130);
        g.drawString(answerStr, 70, 150);
    }

    public int CalcAnswer() throws NumberRangeException
    {
        int int1, int2;
        int answer = -1;
        String str1 = textField1.getText();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        String str2 = textField2.getText();
        try
        {
            int1 = Integer.parseInt(str1);
            int2 = Integer.parseInt(str2);

                //check to make sure both integers are within the range
            if ((int1 < 10) || (int1 > 20) ||
                (int2 < 10) || (int2 > 20))
            {
                //since they are not, throw our custom exception
                NumberRangeException e = new NumberRangeException
                    ("Numbers not within the specified range.");
                throw e;
            }
            answer = int1 + int2;
        }
        catch (NumberFormatException e)
        {
            answerStr = e.toString();
        }
        return answer;
    }
}
```

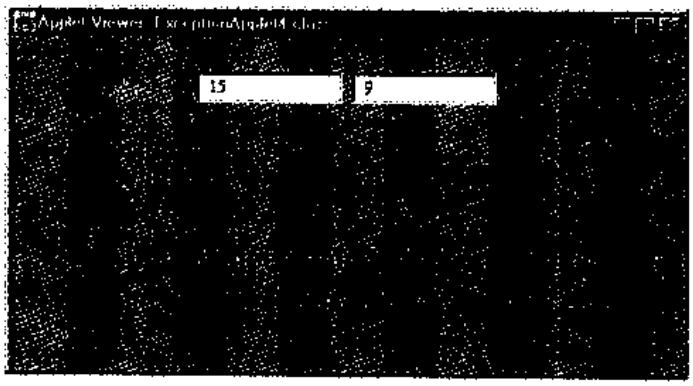


图 20.10　小应用程序捕捉 NumberRangeException 异常

在 ExceptionApplet4 小应用程序的 action( )方法中,程序调用了本地的 CalcAnswer( )方法。因为 CalcAnswer( )方法抛出一个 NumberRangeException 异常(用户刚刚创建的类),所以在 action( )方法的 try 和 catch 程序块中必须包括这个方法的调用。在 CalcAnswer( )方法中,程序获取用户在文本框中输入的内容并把返回的字符串转换成整数。因为 parseInt( )方法调用可以抛出 NumberFormatException 异常,所以 CalcAnswer( )在 try 程序块中包含了 parseInt( )方法调用。在 try 块中程序不仅把字符串转换成整数,而且还检查该整数是否落在

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

适当的范围内。如果它们超出了这个范围，程序就创建并抛出一个 NumberRangeException 类的对象。

20.2 Java 的 Error 类

迄今为止，用户已经看过了一些可以在用户的程序中处理的异常类。Java 还定义了错误类，Error 类实际上是特殊的异常类。和 Exception 类一样，Error 类也是从 Throwable 类派生出来的。但是也有一些代表严重错误的特殊错误类是从 Error 类派生出来的，比如内部错误和与类有关的问题，这类错误程序不允许用户插手处理，Java 系统会处理这些错误。

下面的错误是按照它们的继承层次排列的。类名后面括号里是定义该类的包（除了一个是在 java.awt 包中定义的外。所有的错误类都是在 java.lang 包中定义的。

Throwable (java.lang)
Error (java.lang)
AWTError (java.awt)
ThreadDeath (java.lang)
LinkageError (java.lang)
ClassCircularityError (java.lang)
ClassFormatError (java.lang)
NoClassDefFoundError (java.lang)
UnsatisfiedLinkError (java.lang)
VerifyError (java.lang)
IncompatibleClassChangeError (java.lang)
AbstractMethodError (java.lang)
IllegalAccessError (java.lang)
InstantiationError (java.lang)
NoSuchFieldError (java.lang)
NoSuchMethodError (java.lang)
VirtualMachineError (java.lang)
InternalError (java.lang)
OutOfMemoryError (java.lang)
StackOverflowError (java.lang)
UnknownError (java.lang)

20.3 Java 的事件

正如用户已经知道的那样，事件用于描述程序、系统和程序使用者之间的各种活动。当用户使用程序做某些事情时，比如在程序的窗口中单击鼠标，这时系统会创建一个代表这一动作的事件并将该事件传递给程序的事件处理代码。由这段代码决定如何处理该事件，这样用户就可以得到适当的反应。

例如，当程序的使用者单击某个按钮时，他希望与该按钮有关的命令得到执行。通过阅读第 15 章“高级小应用程序代码”用户可以了解如何在小应用程序中使用事件。现在让我们通

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

过研究处理事件的类来深入考查 Java 的事件和如何创建与处理事件。

20.4　事件处理技术

Java 有两个不同的事件处理系统。Java 1.1 引入了监听—适配器系统，该系统在 Java 1.2 中得到了延用。在本章的后面部分将介绍 Java 1.0 事件模式，在这种模式下，使用 Component 类处理事件。这意味着任何处理 Java 1.0 中需要处理事件的类必须追溯它的祖先到 Component 类。版本 1.1 更新了事件模式，在 Java 1.1 中任何类都可以接收和管理事件，不管该类是否以 Component 类为超类。

在 Java 1.1 中事件经常是由另外的对象处理的。例如，当用户单击按钮时，用户希望在按钮之外的对象中处理该动作事件。为了支持这种处理方法，事件模式必须支持在 java.util.EventListener 中定义的事件监听器。

事件监听器是指定实现一个或多个监听器接口的任意一个对象。对于每一个 AWT 事件类型都有不同的监听器。例如，MouseListener 接口定义了诸如 mouseClicked、mousePressed 和 mouseReleased 之类的方法类。为了接收来自部件的事件，需要在其中增加一个用于监听部件的事件的对象。如果某个对象实现了 MouseListener 接口，那它通过在该部件上调用 addMouseListener 方法监视部件的鼠标事件。这样可以允许用户不经过创建部件的子类就可以处理部件的事件，也不必在父容器中处理事件。

MusicalButton 对象实现了 ActionListener 接口，它通过 actionPerformed 方法接收 ActionEvent 对象。因为 MusicalButton 不再是按钮类的子类，所以用户可以将它挂在任何类型的按钮上，无需增加额外的代码。所有的按钮都支持添加监听器的 addActionListener 方法。

Java 1.1 事件模式中的各种类型的监听器也都有适配器对象。适配器是非常简单的对象，它实现了特定的接口，在接口中为各种方法定义了空的体。如果用户定义了一个专门实现 KeyListener 接口的对象，那么用户只创建 KeyAdapter 子类的对象并根据用户的喜好覆盖某些方法即可。对于所有的接口用户都可以这样做。

在 Java 1.1 事件模式中有一个大的 Event 对象，该对象包含了所有可能发生的事件的所有属性。而在 Java 1.2 中是由不同的事件对象处理不同的事件。键盘事件被传递给 KeyEvent 对象，动作被传递给 ActionEvent 对象。这样做可以使事件相对小一些，因为它不需要包含事件所有可能的变体。

警告：尽管在当前版本中仍然支持 Java 1.0 事件，但是用户不要在同一个程序中混合使用 1.0 事件模式和 1.1 事件模式。难以保证它们能同时正常工作。

20.4.1　事件监听器

在 Java 1.2 中，事件是由事件监听器管理的。事件监听器是在 Java 系统中注册的用于接收特殊事件的类。只有被事件监听器注册过的事件才能够被该监听器接收到。

在本部分中，用户可以修改 EventApplet 小应用程序，使得它可以使用 Java 1.1 事件模式。在前一部分中开发的该程序的原版本使用的是类似于 Java 1.0 的事件模式。

把 EventApplet 修改成 EventApplet2 的第一步是在源代码文件的顶部增加下面这行语句：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.awt.event.*;
```

该行代码允许程序访问在事件包中定义的新类。如果不包括这个包，那么使用 Java 1.1 事件模式的程序就不能够通过编译。

第二步，必须确定用户的程序要处理的事件和哪个部件产生这些事件。为了使事情简单明了，我们编写的 EventApplet2 只响应 action 事件，action 事件是小应用程序处理的最普通的事件。表 20.2 提供了用户把小应用程序转换为 Java 1.1 事件模式需要的有关信息。该表列出了所有的事件，显示了产生这些事件的部件和 Java1.1 事件模式使用的接口。

表 20.2 Java 1.1 事件模式概述

事　　件	部　　件	接　　口
ACTION_EVENT	按钮、列表框、菜单项	ActionListener
ACTION_EVENT	复选框、选择框	ItemListener
ACTION_EVENT	文本域	ActionListener
GOT_FOCUS	部件	FocusListener
KEY_ACTION	部件	KeyListener
KEY_ACTION_RELEASE	部件	KeyListener
KEY_PRESS	部件	KeyListener
KEY_RELEASE	部件	KeyListener
LIST_DESELECT	复选框、复选框菜单项	ItemListener
LIST_DESELECT	选择框、列表框	ItemListener
LIST_SELECT	复选框、复选框菜单项	ItemListener
LIST_SELECT	选择框、列表框	ItemListener
LOST_FOCUS	部件	FocusListener
MOUSE_DOWN	画布、对话框、框架	MouseListener
MOUSE_DOWN	面板、窗口	MouseListener
MOUSE_DRAG	画布、对话框、框架	MouseMotionListener
MOUSE_DRAG	面板、窗口	MouseMotionListener
MOUSE_ENTER	画布、对话框、框架	MouseListener
MOUSE_ENTER	面板、窗口	MouseListener
MOUSE_EXIT	画布、对话框、框架	MouseListener
MOUSE_EIXT	面板、窗口	MouseListener
MOUSE_MOVE	画布、对话框、框架	MouseMotionListener
MOUSE_MOVE	面板、窗口	MouseMotionListener
MOUSE_UP	画布、对话框、框架	MouseListener
MOUSE_UP	面板、窗口	MouseListener
SCROLL_ABSOLUTE	滚动条	AdjustmentListener
SCROLL_BEGIN	滚动条	AdjustmentListener
SCROLL_END	滚动条	AdjustmentListener
SCROLL_LINE_DOWN	滚动条	AdjustmentListener

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

事　　件	部　　件	接　　口
SCROLL _ LINE _ UP	滚动条	AdjustmentListener
SCROLL _ PAGE _ DOWN	滚动条	AdjustmentListener
SCROLL _ PAGE _ UP	滚动条	AdjustmentListener
WINDOW _ DEICONIFY	对话框、框架	WindowListener
WINDOW _ DESTROY	对话框、框架	WindowListener
WINDOW _ EXPOSE	对话框、框架	WindowListener
WINDOW _ ICONIFY	对话框、框架	WindowListener
WINDOW _ MOVED	对话框、框架	ComponentListener

从这个表中可以看到，由按钮部件产生的 ACTION _ EVENT 事件应该由 ActionListener 接口处理。这意味着 EventApplet2 必须声明实现了 ActionListener 接口，例如：

```
public class EventApplet2 extends Applet implements ActionListener
```

现在为了从按钮事件中接收事件，必须在程序中创建一个按钮，并将该按钮注册为 ActionListener，然后将按钮加进小应用程序中。所有这些工作可以在 init ( )方法中完成，比如：

```
button1 = new Button ("Test Button")
button1.addActionListener(this);
add(button1);
```

正如用户看到的那样，现在的 Button 类拥有了名为 addActionListener( )的方法，该方法负责将按钮注册为 ActionListener。其它的部件也有类似的新方法。例如，现在的滚动条类拥有了名为 addAjustmentListener( )方法，该方法负责将滚动条注册为 AdjustmentListener。

因为小应用程序实现了 ActionListener 接口，所以它必须实现接口中的每个方法。幸运的是 ActionListenr 只声明了一个方法：actionPerformed( )方法。该方法替换了程序处理动作事件的老 action( )方法。程序清单 20.11 显示了老的 action( )方法，程序清单 20.12 显示了新的 actionPerformed( )方法。

```
public boolean action(Event evt, Object arg)
{
    if (arg == "Test Button")
    {
        if (color == Color.black)
            color = Color.red;
        else
            color = Color.black;
        repaint();
        return true;
    }
    return false;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
}
```

清单清单 20.12 ex20_17.txt——新的 actionPerformed 方法

```
public void actionPerformed(ActionEvent event)
{
    String arg = event.getActionCommand();
    if (arg == "Test Button")
    {
        if (color == Color.black)
            color = Color.red;
        else
            color = Color.black;
        repaint();
    }
}
```

如果用户仔细考虑这两个程序清单就会发现它们有两点不同。首先,action( )方法返回布尔值,而 actionPerformed( )方法没有返回值。第二,按钮标签的 arg 变量被当作参数传递给了 action( )方法来获得标签文本。程序清单 20.13 显示了完整的 EventApplet2,它使用 Java 1.1事件模式处理按钮部件。

[redacted]

```
import java.awt.event.*;
import java.awt.*;
import java.applet.*;

public class EventApplet2 extends Applet
    implements ActionListener
{
    Button button1;
    Color color;

    public void init()
    {
        button1 = new Button("Test Button");
        button1.addActionListener(this);
        add(button1);
        color = Color.black;
        resize(400, 200);
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void paint(Graphics g)
    {
        Font font = new Font("TimesRoman", Font.PLAIN, 48);
        g.setFont(font);
        g.setColor(color);
        g.drawString("TEST COLOR", 55, 120);
    }

    //* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    // Here is the method implementation for the
    // ActionListener interface.
    //* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    public void actionPerformed(ActionEvent event)
    {
        String arg = event.getActionCommand();
        if (arg = = "Test Button")
        {
            if (color = = Color.black)
                color = Color.red;
            else
                color = Color.black;
            repaint();
        }
    }
}
```

注意：为了确定监听器接口的哪一个方法必须在用户的程序中实现，可以加载接口的源代码并将方法声明拷贝到用户自己的程序中。然后编写要使用的接口方法的代码并把剩下的声明改成空的方法。

20.5　键盘和鼠标事件

让我们继续介绍 Java 1.1 事件模式。正如用户已经了解的那样，用户可以接收到下列键盘事件和鼠标事件：键被按下，被释放，鼠标进入小应用程序窗口，鼠标离开小应用程序窗口；鼠标键被按下，鼠标键被释放；鼠标被移动和拖动（按住鼠标键的同时移动鼠标）。

20.5.1　键盘事件

要想在 Java 1.2 事件模式下监听键盘事件，用户需要实现 KeyListener 接口。KeyListener 接口包含三个方法：keyPressed，keyReleased 和 keyTyped。当键被按下时调用 keyPressed 方

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

法，当键被释放时调用 keyReleased 方法。keyTyped 方法是 keyPressed 和 keyReleased 的组合。当键被按下后又释放时（比如普通的键盘输入），keyTyped 方法就会被调用。下面是 keyListener 接口的方法声明：

```
public abstract void keyTyped(KeyEvnet event)
public abstract void keyPressed(KeyEvnet event)
public abstract void keyReleased(KeyEvnet event)
```

每个键盘事件都有一个关联的键码，该键码可以通过在 KeyEvnet 中调用 getKeyCode 方法获得：

```
pbulic int getKeyCode( )
```

键码可以是输入的字符，可以是普通的字符，也可以是特殊的键（功能键、光标移动键、键盘控制键等等）。某些键是与活动有关的键。动作键（带箭头的键，Home 键，End 键），功能键是 F1 到 F12 键、打印屏幕键（PrintScreen）和锁定键（大小写锁定键 Caps Lock，数字锁定键 Num Lock 和滚动锁定键 Scroll Lock）。在 KeyEvent 中的 isActionKey 方法当键涉及的是动作时返回 true 值：

```
public boolean isActionKey( )
```

因为键码是取决于系统的，所以 AWT 为普通的键定义了自己的码。在 KeyEvent 类中定义的键码如下面的表所示。注意表中的值可以是单键值，也可以是组合键的值。

键　　码	键
KeyEvent.F1 – KeyEvent.F12	功能键 F1 – F12
KeyEvent.VK_LEFT	向左箭头键
KeyEvent.VK_RIGHT	向右箭头键
KeyEvent.VK_UP	向上箭头键
KeyEvent.VK_DOWN	向下箭头键
VK_KP_UP	小键盘的向上箭头键
VK_KP_DOWN	小键盘的向下箭头键
VK_KP_LEFT	小键盘的向左箭头键
VK_KP_RIGHT	小键盘的向右箭头键
VK_END	END 键
VK_HOME	Home 键
VK_PAGE_DOWN	向后翻页键
VK_PAGE_UP	向前翻面键
VK_PRINTSCREEN	打印屏幕键
VK_SCROLL_LOCK	滚动锁定键
VK_CAPS_LOCK	大写锁定键
VK_NUM_LOCK	数字锁定键
PAUSE	暂停键

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

键　　码	键
VK_INSERT	插入键
VK_DELETE	删除键
VK_ENTER	回车键
VK_TAB	制表符键
VK_BACK_SPACE	退格键
VK_ESCAPE	Esc 键
VK_CANCEL	取消键
VK_CLEAR	清除键
VK_SHIFT	shift 键
VK_CONTROL	Ctrl 键
VK_ALT	Alt 键
VK_PAUSE	暂停键
VK_SPACE	空格键
VK_COMMA	逗号键
VK_SEMICOLON	分号键
VK_COLON	冒号键
VK_AMPERSAND	& 键
VK_EXCLAMATION_MARK	! 键
VK_INVERTED_EXCLAMATION_MARK	¡键
VK_AT	@键
VK_CIRCUMFLEX	^ 键
VK_NUMBER_SIGN	# 键
VK_DOLLAR	$ 键
VK_UNDERSCORE	_键
VK_PLUS	+ 键
VK_EURO_SIGN	欧洲货币常量键
VK_EQUALS	= 键
VK_LESS	< 键
VK_GREATER	> 键
VK_PERIOD	. 键
VK_SLASH	/ 键
VK_BACK_SLASH	\ 键
VK_0 - VK_9	0 - 9 键
VK_A - VK_Z	A - Z 键(等于 ASCⅡ 值)
VK_OPEN_BRACKET	[键
VK_CLOSE_BRACKET	] 键
VK_BRACELEFT	{ 键

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

键　　码	键
VK_BRACERIGHT	} 键
VK_LEFT_PARENTHESIS	(键
VK_RIGHT_PARENTHESIS	) 键
VK_NUMPAD0 - VK_NUMPAD9	小键盘上的 0 ~ 9 键
VK_MULTIPLY	多键
VK_ADD	加键
VK_SUBTRACT	减键
VK_DIVIDE	除键
VK_DECIMAL	·键
VK_SEPARATER	分隔符键
VK_HELP	帮助键
VK_META	Meta 键
VK_QUOTE	单引号‘ 键
VK_BACK_QUOTE	单引号 ’键
VK_QUOTEDBL	“ 键
VK_CUT	剪切键
VK_COPY	拷贝键
VK_PASTE	粘贴键
VK_UNDO	取消键
VK_AGAIN	重复键
VK_FIND	查找键
VK_PROPS	属性键
VK_STOP	停止键
VK_COMPOSE	排版键
VK_ALT_GRAPH	图形转换键
VK_UNDEFINED	无键码值键

注意:在 JDK 1.2 的 KeyEvent 中的许多常量都改变了它们的名字,如果用户使用的是以前的 JDK,就需要引用 API 去获得正确的值。

因为许多键码就是普通的字符,所以用户可以使用 getKeyChar( )来获得击键产生的字符:

```
public char getKey Char( )
```

20.5.2 Java 2 中的修饰键

用户可能认为修饰键(Control 键、Alt 键、Shift 键、Meta 键)也是键盘事件,但实际上它们不是键盘事件。在大部分窗口化的操作系统中,用户可以把这些键与鼠标结合起来使用。在

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java 1.2 事件层次中包含 InputEvent 类，该类是 KeyEvent 类和 MouseEvent 类的超类。InputEvent 类里的 getModifiers( )方法返回一个位图，该位图指示当事件发生时，哪个修饰键处于激活状态：

```
public int getModifiers( )
```

用户可以使用 InputEvent 的 SHIFT _ MASK，CTRL _ MASK，META _ MASK 和 ALT _ MASK 属性检查由 getModifiers( )返回的位。例如，下面的代码片段检查事件发生时 Alt 键是否被按下：

```
InputEvent evt;
if ((evt.getModifiers( )& InputEvent.ALT _ MASK)! =0){
//Alt 键被按下
}
```

因为这种检查修饰符的方法比较麻烦，所以 InputEvent 类还定义了另外的捷径：

```
public boolean isShiftDown( )
public boolean isControlDown( )
public boolean isMetaDown( )
```

鼠标按钮也可以认为是修饰键。InputEvent 类的 BUTTON1 _ MASK，BUTTON2 _ MASK 和 BUTTON3 _ MASK 属性允许用户在事件发生时检查鼠标按钮是否被按下。但是这三个方法没有捷径。

除了修饰信息以外，用户还可以通过调用 getWhen( )获得输入事件发生的时间，getWhen( )返回一个类似于 System.currentTimeMillis 返回的时间：

```
public long getWhen( )
```

20.5.3　Java 2 中的鼠标事件

在 Java 2 中有两个不同的监听器接口用于监听鼠标事件。在大部分情况下，用户只需要使用 MouseListener 接口就足够了，MouseListener 接口用于定义与鼠标移动无关的方法。

mousePressed 方法和 mouseReleased 方法指示鼠标按钮的按下或释放：

```
public abstract void mousePressed(MouseEvent event)
public abstract void mouseReleased(MouseEvent event)
```

如果用户不希望跟踪按钮被按下和被释放的时间，用户可以使用 mouseClicked( )方法，当鼠标按钮被按下后又被释放时调用该方法：

```
public abstract void mouseClicked(MouseEvent event)
```

MouseEvent 对象的 getClickCount( )方法告诉用户鼠标被点击的次数，这样用户可以检测鼠标是否被双击：

```
public int getClickCount( )
```

当鼠标进入部件区域时调用 mouseEntered( )方法，当鼠标从部件区域离开时调用 mouseExited( )方法：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public abstract void mouseEntered (MouseEvent event)
public abstract void mouseExited (MouseEvent event)
```

用户可以通过调用 MouseEvent 对象中的 getPoint( )方法获得当事件发生时鼠标的 x,y 坐标,也可通过调用 getX( )方法和 getY( )方法来分别获得:

```
public synchronized Point getPoint( )
public int getX( )
public int getY( )
```

因为大部分应用程序不需要跟踪鼠标的移动,所以鼠标移动的方法被放置在一个单独的接口中了。这样可以使得用户可以不必每次都要获得一个事件才可以监听鼠标被按下的动作。MouseListener 接口实现了两个跟踪鼠标移动的方法。当鼠标被移动但鼠标按钮未被按下时调用 mouseMove( )方法,当按住鼠标按钮移动鼠标时,Java 调用 mouseDragged( )方法:

```
public abstract void mouseMoved (MouseEvent event)
public abstract void mouseDragged (MouseEvent event)
```

20.6 Java 1.0 的事件模式

在 JDK 1.1 以前的事件模式与用户在本章前面看到的几乎完全不同。那个模式在 JDK 1.1和JDK 1.2中得到了维护,但不再使用(这意味着将来有可能删除它)。不幸的是,较老一点的浏览器(Netescape2.0～4.0 和 Internet Explorer2.0～3.0)都不支持 Java1.1 事件模式。由于某种原因经常需要编写 1.0 模式的代码,所以我们接下来有必要介绍一下它。

20.6.1 Event 类

在 Java 1.0 中,事件实际上就是 Event 类,该类定义了所有在程序中可能用到的事件,还定义了获取关于事件的信息的方法。Event 类应该说是用户见过的最复杂的类。

Event 类做的第一件事情是为代表事件(比如按键事件)和修饰事件(比如按住 Shift 键单击鼠标按钮)的键定义常量。表 20.3 列出了这些常量及其描述。

表 20.3 在 Java 1.0 事件模式的 Event 类中的键盘常量

常　　量	键
ALT_MASK	ALT(替换)键
CTRL_MASK	CTRL 键
DOWN	向下箭头键
END	END 键
F1	F1 键
F10	F10 键
F11	F11 键
F12	F12 键
F2	F2 键

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

常　　量	键
F3	F3 键
F4	F4 键
F5	F5 键
F6	F6 键
F7	F7 键
F8	F8 键
F9	F9 键
HOME	HOME 键
LEFT	向左箭头键
META_MASK	Meta 键
PGDN	向后翻页键
PGUP	向前翻页键
RIGHT	向右箭头键
SHIFT_MASK	shift 键
UP	向上箭头键

接下来要做的事情是为在 Java 程序中要处理的事件定义常量。这些事件包括从鼠标事件和键盘事件到由移动、最小化和关闭窗口引起的所有事件。表 20.4 列出了这些事件常量，这些常量被用做 Event 对象的 ID。

表 20.4　在 Java 1.0 事件模式的 Evnet 类中的事件常量

常　　量	描　　述
ACTION_EVENT	在支持 action( )的方法中使用
GOT_FOCUS	当窗口(或部件)获得输入焦点时产生
KEY_ACTION	类似于 KEY_PRESS
KEY_ACTION_RELEASE	类似于 KEY_RELEASE
KEY_EVET	普通的键盘事件
KEY_PRESS	当键被按下时产生
KEY_RELEASE	当键被释放时产生
LIST_DESELECT	当释放列表框中的项目时产生
LIST_EVENT	普通的列表框事件
LIST_SELECT	当选择列表框中某个项目时产生
LOAD_FILE	当加载文件时产生
LOST_FOCUS	当窗口(或部件)失去焦点产生
MISC_EVENT	有各种特点的事件
MOUSE_DOWN	当鼠标按钮被按下时产生
MOUSE_DRAG	当鼠标指针被拖动时产生

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

常　量	描　述
MOUSE_ENTER	当鼠标指针进入窗口时产生
MOUSE_EVENT	普通的鼠标事件
MOUSE_EXIT	当鼠标指针离开窗口时产生
MOUSE_MOVE	当鼠标移动时产生
MOUSE_UP	当鼠标按钮被释放产生
SAVE_FILE	当文件被存储时产生
SCROLL_ABSOLUTE	当移动滚动条的滑块按钮时产生
SCROLL_EVENT	普通的滚动事件
SCROLL_LINE_DOWN	当单击滚动条的向下箭头时产生
SCROLL_LINE_UP	当单击滚动条的向上箭头时产生
SCROLL_PAGE_DOWN	当单击滚动条的滑块按钮与向下箭头之间的区域时产生
SCROLL_PAGE_UP	当单击滚动条的滑块按钮与向上箭头之间的区域时产生
WINDOW_DEICONIFY	当窗口被恢复时产生
WINDOW_DESTROY	当窗口被撤销时产生
WINDOW_EVENT	普通的窗口事件
WINDOW_EXPOSE	当窗口被露出来时发生
WINDOW_ICONIFY	当窗口被最小化时产生
WINDOW_MOVED	当窗口被移动时产生

像大多数类一样，Event 类也声明了大量的数据成员用以存储关于事件对象信息。当响应某个事件时用户可能要检查其中的一个或多个数据成员。例如，当响应大多数鼠标事件时，用户通常希望知道当事件发生时鼠标所在位置的 x，y 坐标值。表 20.5 列出了这些数据成员和对它们的描述。

表 20.5　Event 类的数据成员

数　据　成　员	描　　述
arg	关于事件的附加信息
clickCount	与事件关联的鼠标单击次数
evt	在列表中的下一个事件
id	事件的 ID(参见表 20.4)
key	发生键盘事件的键
keyChar	被按下的字符键
modifiers	事件的修饰键(参见表 20.3)
target	产生事件的部件
when	事件发生的时间
x	事件的 x 坐标
y	事件的 y 坐标

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

最后，但不是最不重要的，Event 类定义了许多用于获取关于事件的信息的方法。表 20.6 列出了这些方法及其描述。

表 20.6　Event 类的方法

方　　法	描　　述
controlDown( )	获取 Ctrl 键的状态
metaDown( )	获取 Meta 键的状态
paramString( )	获取事件的参数字符串
ShiftDown( )	获取 Shift 键的状态
toString( )	获取代表对象状态的字符串
translate( )	翻译事件，事件的 x 和 y 坐标发生增减

20.6.2　事件的起源

用户可能会奇怪事件是从哪是来到用户的程序中的。操作系统，比如 Microsoft Windows 或 Micintosh 的 System 7 都跟踪发生在系统中的所有事件。系统把这些事件安排给适当的目标对象。例如，如果用户单击用户的小应用程序窗口，系统将构造一个鼠标被按下的事件并发送给窗口处理。由窗口选择用该事件做件什么事情或者将事件传回系统做缺省的处理。

在 Java 1.0 事件模式中，当 Java 截获 Java 部件的事件时，经过翻译后安排给适当的对象去处理。因为所有的事件处理实体都取决于当前使用的窗口系统，使用 Java 在由 java.awt 包中定义的类处理这些事件。特别是 Component 类，它负责接收和处理所有从 Component 类派生出来的类的事件。因为在 Java 1.0 小应用程序和应用程序中虚拟的各种可视对象（比如按钮、面板、文本框、画布等等）都可以跟踪它的祖先到 Component 类，所以 Component 类是所有事件处理的老祖宗。由于这些原因，Component 类定义了许多与事件有关的方法。表 20.7 列出了这些方法和对这些方法的描述。

表 20.7　Component 类的事件处理方法

方　　法	描　　述
action( )	响应有动作事件的部件
deliverEvent( )	向部件发送事件
handleEvent( )	安排事件给适当的处理者去处理
keyDown( )	响应键被按下事件
keyUp ( )	响应键被释放事件
mouseDown( )	响应鼠标按钮被按下事件
mouseDrag( )	响应鼠标被拖动事件
mouseEnter( )	响应鼠标进入事件
mouseExit ( )	响应鼠标退出事件
mouseMove ( )	响应鼠标移动事件
mouseUp( )	响应鼠标按钮被释放事件
postEvent ( )	类似于 deliverEvent

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:如果用户在 Java 1.0 事件模式下使用 JDK 1.1 或 JDK 1.2,编译器将发出警告信息:这些方法不再被使用了。这些警告并不是实际的问题,但是它让用户知道这些方法将来将不被支持。

在 Component 类中,事件处理方法,比如 action( )、mouseDown ( )和 keyDown 除了返回 false 外不做任何实际的事情,当返回 false 时是向 Java 指明该事件尚未被处理。这些方法必须在用户的程序中覆盖掉方能使程序以适当的处理响应该事件。例如,如果用户在小应用程序中没有覆盖 mouseDown( )方法,那么缺省的 mouseDown( )版本将返回 false,该值告诉 Java 该信息需要在下面的行中进一步处理。在鼠标被按下的事件中,Java 可能返回未处理的事件给系统,由系统作缺省地处理(这意味着该事件将被忽略掉)。

程序清单 20.14 中的小应用程序用打印单词 Click! 来响应鼠标单击,只要用户在小应用程序中单击鼠标就会打印出单词 Click!。它覆盖了 mouseDown( )方法并使用小应用程序的 coordX 和 coordY 数据域存储鼠标单击时的位置。程序清单 20.14 显示了运行在小应用程序浏览器中的 MouseApplet 的全部代码。

```
import java.awt.*;
import java.applet.*;
public class MouseApplet extends Applet
{
    int coordX, coordY;
    public void init()
    {
        coordX = -1;
        coordY = -1;
        Font font = new Font("TimesRoman", Font.BOLD, 24);
        setFont(font);
        resize(400, 300);
    }
    public void paint(Graphics g)
    {
        if (coordX != -1)
            g.drawString("Click!", coordX, coordY);
    }
    public boolean mouseDown(Event evt, int x, int y)
    {
        coordX = x;
        coordY = y;
        repaint();
        return true;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

}
}

注意:当用户运行 MouseApplet 时,用户可能发现每当 paint( )方法被调用时小应用程序窗口都被擦除。这就是只有一个 Click! 出现在窗口中的原因。

20.6.3　键盘

键盘的出现比鼠标的出现要早,从它成为人类和计算机之间的主要接口到现在已经有几十年了。虽然键盘非常重要,但以往都是在较低层次上处理键盘事件,不像处理 TextField 部件那样概略。Java 有两个与键有关的事件,这两个事件使用 KEY _ PRESS 和 KEY _ RELEASE 常量代表。正像用户将要看到的那样,Java 定义了一些方法使得对键盘的响应与对鼠标的响应一样简单。在第 15 章我们介绍过键盘事件;在本部分里我们将进一步介绍如何在用户的 Java 程序中处理键盘事件。

在小应用程序存活期间当用户按下键盘上某个键时,Java 会向小应用程序发送一个 KEY _ PRESS 事件。在用户的 Java 程序中,用户可以通过覆盖 KeyDown( )方法来响应这个事件,该方法的签名类似于:

```
public boolean KeyDown(Event evt, int Key)
```

正像用户看到的那样,这个方法接收两个参数,即 Event 对象和代表被按下的键的整数。这个整数实际就是该键代表的字符的 ASCⅡ码值。为了在程序中使用该值,用户首先需要把它强制成字符值,例如:

```
char c = (char)key;
```

键盘上的许多键并不产生字符,而是发出命令。这些键包括所有的功能键、Shift 键、Ctrl 键、PageUp 键、PageDown 键,等等。为了使这种类型的键在用户的小应用程序中容易处理,Java 的 Event 类定义了一套常量用以代表这些键值(可参考表 20.3)。

Event 类还为修饰键定义了一些常量,修饰键可以与基本键同时被按下。这些常量也全部在表 20.3 中列出,其中包括 ALT _ MASK,SHIFT _ MASK 和 CTRL _ MASK,它们分别代表键盘上的 ALT(替换)键、Shift(上档)键和 Ctrl(控制)键。其中 SHIFT _ MASK 常量和 CTRL _ MASK 常量分别用在 Event 类的 shiftDown( )方法和 controlDown( )方法中,这两个方法都返回布尔型的值,该值指示修饰符键是否被按下(目前还没有 altDown( )方法)。用户也可以通过检查 Event 对象的 modifiers 域来确定是否某个特殊的修饰符键被按下了。例如,如果用户想要检查 Alt 键,用户可以使用类似下一行语句的 Java 代码来查询:

```
boolean altPressed = (evt.modifiers & Event.ALT _ MASK)! =0;
```

通过使用修饰符域和掩码的逻辑与运算,用户可以得知 Alt 键是否被按下;如果逻辑运算的结果非零则说明 Alt 键被按下了,如果为零则说明 Alt 键未被按下。用户通过将逻辑运算的值与零进行比较来产生一个布尔值。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

20.6.4 直接处理事件

用户的小应用程序使用 Java 1.0 事件模式接收的所有事件都是由从 Component 类继承来的 handleEvent( )方法处理的。当该方法未在用户的程序中覆盖时,缺省的实现负责调用多种响应事件的方法。程序清单 20.15 显示了 handleEvent( )是如何在 Component 类中实现的。通过查看这个程序清单,用户可以很容易地看出为什么必须覆盖 mouseDown( )方法才能响应事件。在接下来的部分用户可以看到如何在自己的程序中自定义 handleEvent 方法。

```
public boolean handleEvent(Event evt) {
switch (evt.id) {
  case Event.MOUSE_ENTER:
    return mouseEnter(evt, evt.x, evt.y);
  case Event.MOUSE_EXIT:
    return mouseExit(evt, evt.x, evt.y);
  case Event.MOUSE_MOVE:
    return mouseMove(evt, evt.x, evt.y);
  case Event.MOUSE_DOWN:
    return mouseDown(evt, evt.x, evt.y);
  case Event.MOUSE_DRAG:
    return mouseDrag(evt, evt.x, evt.y);
  case Event.MOUSE_UP:
    return mouseUp(evt, evt.x, evt.y);
  case Event.KEY_PRESS:
  case Event.KEY_ACTION:
    return keyDown(evt, evt.key);
  case Event.KEY_RELEASE:
  case Event.KEY_ACTION_RELEASE:
    return keyUp(evt, evt.key);

  case Event.ACTION_EVENT:
    return action(evt, evt.arg);
  case Event.GOT_FOCUS:
    return gotFocus(evt, evt.arg);
  case Event.LOST_FOCUS:
    return lostFocus(evt, evt.arg);
}
return false;
}()
```

20.6.5 覆盖 handleEvent( )方法

虽然缺省的 handleEvent( )方法可以调用用户在自己的程序中为每个事件覆盖的特殊方

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

法，但是用户可能希望将所有的事件处理方法都集成在一个方法内单独保存起来，籍此改变小应用程序响应某个特殊事件的方式，或者创建自己的事件。要想完成这些任务中的某些部分（或其它的用户提出的方法），用户可以忽略单个的事件处理方法并覆盖 handleEvent( )方法。

在用户自己的 handleEvent 版本中，用户必须检查 Event 对象的 id 域，目的是确定哪个事件正在被处理。用户可以忽略掉自己不感兴趣的方法。当用户忽略某个事件必须保证它返回 false 值，这样可以使得 Java 知道应该将它向上一个对象层次传递该事件。程序清单 20.16 显示一个为了响应某些事件而覆盖了 handleEvent( )方法的小应用程序。

```
import java.awt.*;
import java.applet.*;
public class DrawApplet2 extends Applet
{
    Point startPoint;
    Point points[];
    int numPoints;
    boolean drawing;
    public void init()
    {
        startPoint = new Point(0, 0);
        points = new Point[1000];
        numPoints = 0;
        drawing = false;
        resize(400, 300);
    }
    public void paint(Graphics g)
    {
        int oldX = startPoint.x;
        int oldY = startPoint.y;
        for (int x = 0; x < numPoints; ++x)
        {
            g.drawLine(oldX, oldY, points[x].x, points[x].y);
            oldX = points[x].x;
            oldY = points[x].y;
        }
    }
    public boolean handleEvent(Event evt)
    {
        switch(evt.id)
        {
            case Event.MOUSE_DOWN:
                drawing = true;
                startPoint.x = evt.x;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                startPoint.y = evt.y;
                return true;
            case Event.MOUSE_MOVE:
                if ((drawing) && (numPoints < 1000))
                {
                    points[numPoints] = new Point(evt.x, evt.y);
                    ++numPoints;
                    repaint();
                }
                return true;
    default:
                return false;
        }
    }
}
```

注意:在程序清单 20.16 中,程序为了能够处理低级的事件覆盖了 handleEvent( )方法。这种技术的一个副作用是除了这些被显式地处理的事件以外的事件都被忽略了。如果用户希望仍然能够正常地响应所有的其它事件,用户必须保证在自己的 handleEvent( )版本中包含它们,最简单的办法就是在用户的新版本中调用原版本的 handleEvent( )方法,具体作法是在返回 false 处使用 super. handleEvent(evt)语句。

20.6.6 发送用户自己的事件

由 Java 创建并分发的事件可能会不完全适合用户程序的需要。在这种情况下,用户可以创建和发送自己的事件。例如,用户可能希望通过按键和单击按钮都可以选取某个命令。处理这种需求的一种办法是在用户的 action( )方法中和 keyDown( )方法中放置几乎完全相同的代码。在 action( )方法中的代码处理单击按钮事件,在 keyDown( )方法中的代码处理按键事件,如程序清单 20.17 所示。

```
public boolean action(Event evt, Object arg)
{
    if (arg == "Test Button")
    {
        if (color == Color.black)
            color = Color.red;
        else
            color = Color.black;
        repaint();
        return true;
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        return false;
    }
    public boolean keyDown(Event evt, int key)
    {
        if ((key == LOWERCASE_T) || (key == UPPERCASE_T))
        {
            if (color == Color.black)
                color = Color.red;
            else
                color = Color.black;
            repaint();
            return true;
        }
        return false;
    }
```

在程序清单 20.18 中显示了一个该问题的更完美的解决办法。在该程序中用户可以创建自己的事件，用它来响应按键和向按钮部件传送事件。用户可以通过调用 Event 类的构造器创建自己的事件，例如：

```
Event event = new Event(button1, Event.ACTION_EVENT, "Test Button");
```

这三个参数是事件的目标部件、事件的 ID 和额外的与事件类型有关的适当信息。对于按钮动作事件来说，第三个参数应该是按钮的标签。

当用户构造完事件以后，用户可以通过调用 deliverEvent( )方法很容易地将该事件发送出去，例如：

```
deliveEvent(event);
```

这个方法的唯一参数是用户想发送的 event 对象。

程序清单 20.18 中的小应用程序为了把按键事件与单击按钮事件联系起来创建并发送了自己的事件。在这个小应用程序中，当用户单击按钮时，文本的颜色发生改变。当用户按下键盘上的 T 键时，文本的颜色也发生改变。这是因为 keyDown( )方法监视 T 键的按动(不论大写还是小写)。当 keyDown 得到 T 键被按下的事件时，程序创建一个 ACTION_EVENT 事件并将它发送出去。这将引起 Java 调用带有事件参数的 action( )方法，与用户单击鼠标按钮的效果完全一样。图 20.11 显示了在小应用程序浏览器下运行的 EventApplet。

```
import java.awt.*;
import java.applet.*;
public class EventApplet extends Applet
{
    Button button1;
    String str;
    Color color;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
final int LOWERCASE_T = 116;
final int UPPERCASE_T = 84;
public void init()
{
    button1 = new Button("Test Button");
    add(button1);
    str = "TEST COLOR";
    color = Color.black;
    resize(400, 200);
}
public void paint(Graphics g)
{
    Font font = new Font("TimesRoman", Font.PLAIN, 48);
    g.setFont(font);
    g.setColor(color);
    g.drawString(str, 55, 120);
}
public boolean action(Event evt, Object arg)
{
    if (arg == "Test Button")
    {
        if (color == Color.black)
            color = Color.red;
        else
            color = Color.black;
        repaint();
        return true;
    }
    return false;
}
public boolean keyDown(Event evt, int key)
{
    if ((key == LOWERCASE_T) || (key == UPPERCASE_T))
    {
        Event event = new Event(button1,
            Event.ACTION_EVENT, "Test Button");
        deliverEvent(event);
        return true;
    }
    return false;
}
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

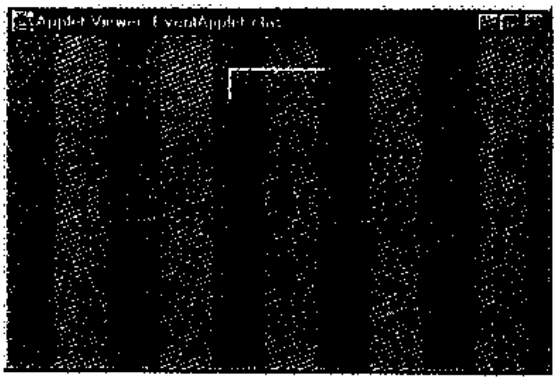


图 20.11　EventApplet 创建和发送自己的事件

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 21 章　容器和布局管理器

本章内容：

21.1　组织部件
21.2　容器
21.3　布局管理器
21.4　再谈容器
21.5　容器基础
21.6　面板
21.7　框架
21.8　对话框
21.9　滚动窗格
21.10　再谈布局管理器
21.11　镶边
21.12　空布局管理器
21.13　Sun 公司的产品进一步扩展

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

21.1　组织部件

AWT 部件实现了用户可以在窗口化的操作系统中可以找到的基本界面小器具。把部件组织在合理的结构是一项比较困难的任务，在 Java 中容器和布局管理器就负责处理这项任务。要想显示部件，用户必须把它放在容器内。例如，小应用程序全部都是容器，因为它们都是 Panel 容器类的子类。

布局管理器像是一个在容器中摆放部件的指令集。当用户在容器内增加一个部件时，容器就会咨询它的布局管理器以找出摆放新部件的位置。旧的技术一般采用绝对坐标来摆放部件，放弃这种习惯是比较困难的，但是用户还是应当适应这种新的模式，因为用户的小应用程序将来可能要在不常见的布局屏幕上运行。最好是将位置摆放工作交给容器和布局管理器来做。

21.2　容器

创建好的用户界面不仅仅是使用部件，部件应当被组织在能够被管理的组中，在这种需求的呼唤下容器就诞生了。容器可以包含部件。除非部件被包含在容器中，否则用户无法使用 AWT 中的部件。没有容器的部件就像一个没有冰箱的冷冻机磁铁一样。在 AWT 中定义的容器有：

- 窗口
- 面板
- 框架
- 对话框

既使用户没有在小应用程序中定义容器，但用户仍然可以使用容器。因为小应用程序的 Applet 类是 Panel 类的子类。

提示：容器不仅可以包含部件，而且它本身也是部件，这意味着一个容器可以包含其它的容器。

21.3　布局管理器

虽然有了可以摆放用户的用户界面(UI)部件的容器，但是用户还需要将部件组织在容器内的方法。这种组织方法就是布局管理器。每个容器都给出了一个布局管理器用于决定显示部件的位置。AWT 中的布局管理器有：

- 流式布局
- 边缘布局
- 网格布局
- 卡片布局
- 网格块布局

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

21.4 再谈容器

除了各种令人惊奇的部件以外，AWT 还提供了几个有用的容器：

- 面板 一个纯粹的容器。它本身不是窗口，它的唯一的用途就是帮助用户在窗口中组织部件。
- 框架 一个功能齐全的窗口，它拥有自己的标题和图标，它可以有下拉式菜单，还可以使用一些不同形状的光标。
- 对话框 一个弹出式的窗口，它的功能不完全与框架一样，对话框主要用来显示一些提示信息，比如“Are you sure you want to quit?”
- 滚动窗格 带有滚动条的窗口，它允许显示比屏幕还大的区域。

21.5 容器基础

所有的容器都执行相同的基本功能，即包含其它的部件。用户可以通过在容器里调用 add 方法在容器内放置部件，例如下面的语句：

```
public synchronized Component add(Component newComponent)
```

该语句将 newComponent 部件加到容器的末尾。容器好像一个数组或一个向量，其中包含的部件都有一个特殊的位置或索引值。请看下面的语句：

```
public synchronized Component add(Component, newComponent, int pos);
```

该语句把 newComponent 部件加在容器的 pos 位置上。从 pos 到末尾的部件全部向后移动。换句话讲，这个方法并不是替换 pos 位置上部件，而是在该位置的部件之前插入一个新部件。

```
public synchronized Component add(String name, Componet newComponet)
```

该语句将 newComponent 部件加在容器的末尾。该部件被作为名为 name 的部件加进了布局管理器。某些布局管理器，比如 BorderLayout 要求每个部件都有一个可见的特殊名字。如果布局管理器不需要它则可以忽略掉。

remove 方法用于删除容器内的部件，例如下面的语句：

```
public synchronized void remove (Component comp);
```

removeAll 方法用于删除容器内的所有部件：

```
public synchronized void removeAll();
```

用户可以用 getComponent 方法获得容器里的第 n 个部件：

```
public sychronized Component getComponent(int n)
throw ArrayIndexOutOfBoundsException;
```

用户可以用 getComponents 方法获取所有的部件：

```
public sychronized Component[ ] getComponents()
```

countComponents 方法返回容器内存放的部件总数：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public int countComponets( )
```

21.6　面板

因为面板只用于组织部件,所以实际可以使用面板做的事情很少。用户可以使用下面的语句创建面板:

```
Panel myPanel = new Panel( )
```

然后用户就可以将该面板加到另外的容器中。例如,用户可以把它加到用户的小应用程序中:

```
add(myPanel);
```

用户也可以嵌套面板,即一个面板包含一个或多个其它的面板:

```
Panel mainPanel , subPanel1 , subPanel2;
subPanel1 = new Panel();//创建第一个子面板
subPanel2 = new Panel();//创建第一个子面板
mainPanel = new Panel();//创建主面板
mainPanel.add(subpanel1);//使 subPanel1 成为 mainPanel 的孩子(子面板)
mainPanel.add(subpanel2);//使 subPanel2 成为 mainPanel 的孩子
```

用户可以嵌套任意多层深度的面板。例如,在前面的例子中用户可以使 subPanel2 成为 subPanel1 的子面板(结果明显地不同)。

程序清单 21.1 显示 了如何创建面板和在它里面嵌套子面板。

```
import java.awt. * ;
import java.applet. * ;

// PanelApplet
//
//.The PanelApplet applet creates a number of panels and
// adds buttons to them to demonstrate the use of panels
// for grouping components.

public class PanelApplet extends Applet
{
    public void init()
    {
// Create the main panels
            Panel mainPanel1 = new Panel();
            Panel mainPanel2 = new Panel();

// Create the sub - panels
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        Panel subPanel1 = new Panel();
        Panel subPanel2 = new Panel();

// Add a button directly to the applet
        add(new Button("Applet Button"));

// Add the main panels to the applet
        add(mainPanel1);
        add(mainPanel2);

// Give mainPanel1 a button and a sub-panel
        mainPanel1.add(new Button("Main Panel 1 Button"));
        mainPanel1.add(subPanel1);

// Give mainPanel2 a button and a sub-panel
        mainPanel2.add(new Button("Main Panel 2 Button"));
        mainPanel2.add(subPanel2);

// Give each sub-panel a button
        subPanel1.add(new Button("Sub-panel 1 Button"));

        subPanel2.add(new Button("Sub-panel 2 Button"));
    }
}
```

图 21.1 显示了 PanelApplet 的输出。

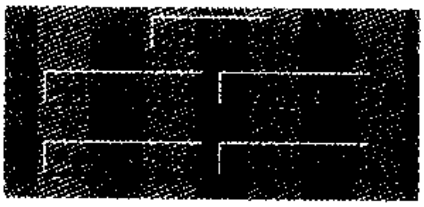


图 21.1 像其它的容器一样，面板可以帮助用户把部件组织在一起

21.7 框架

框架是 AWT 中强大的特性。它允许用户为应用程序创建单独的窗口。例如，有时用户可能希望应用程序运行在网页浏览器之外。用户也可以使用框架建造一个独立的图形应用程序。

21.7.1 创建框架

用户可以使用空构造器创建最初不可见的无标题框架：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

public Frame()

当创建框架时用户可以给出标题，但是该框架仍然是不可见的。

public Frame(String frameTitle)

21.7.2　框架特性

当创建完框架之后，用户可能希望能够看到它。在用户能够看到框架之前，必须给出尺寸。用户可以使用 resize 方法设置框架尺寸：

```
myFrame.resize(300,100);//将框架设成 300 个像素宽,100 个像素高
```

然后可以使用 show()方法使它可见：

```
myFrame.show();//显示框架本身;
```

可以使用 hide 方法隐藏框架。虽然框架不可见，但是它仍然存在：

```
myFrame.hide( );
```

只要框架存在，不管它可见与否都占用运行该框架的窗口系统的资源。如果不再使用某个框架时，用户应该使用 dispose 方法除掉该框架：

public synchronized void dispose( )

用户可以使用 setTitle 方法修改在框架顶部显示的标题：

public void setTitle (String newTitle)

例如：

```
myFrame.setTitle("WithFrame like this, who needs enemies?");
```

getTitle 方法可以返回框架的标题：

public String getTitle()

Frame 类有许多不同的光标。用户可以使用 setCursor 方法修改框架的光标：

public void setCursor(int cursorType)

可以使用的光标有：

Frame.DEFAULT _ CURSOR
Frame.CROSSHAIR _ CURSOR
Frame.TEXT _ CURSOR
Frame.WAIT _ CURSOR
Frame.HAND _ CURSOR
Frame.MOVE _ CURSOR
Frame.N _ RESIZE _ CURSOR
Frame.NE _ RESIZE _ CURSOR
Frame.E _ RESIZE _ CURSOR
Frame.SE _ RESIZE _ CURSOR
Frame.S _ RESIZE _ CURSOR
Frame.SW _ RESIZE _ CURSOR

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Frame.W_RESIZE_CURSOR
Frame.NW_RESIZE_CURSOR

getCursorType 方法可以返回这些值中代表当前光标类型的值：

```
public int getCursorType( )
```

如果用户不希望框架的尺寸被改变，那么用户可以通过调用 setResizable 来转换尺寸设置的开关：

```
public void setResizable(boolean allowResizing)
```

如果框架可以被重新设置尺寸，那么 isResizable 方法返回 true：

```
public boolean isResizable( )
```

用户可以使用 setIconImage 改变框架的图标：

```
setIconImage(Image image)
```

21.7.3 使用框架使用户的小应用程序单独运行

用户可以创建一个既可以作为小应用程序运行又可以作为独立的应用程序来运行的小应用程序。用户需要做的事情是在小应用程序中编写一个 main 方法，在 main 方法中创建一个框架，然后再创建一个属于该框架的小应用程序实例。程序清单 21.2 显示了这样一个小应用程序，该小应用程序既可以作为小应用程序运行，也可以作为独立的应用程序来运行。

```
import java.awt.*;
import java.applet.*;

// StandaloneApplet is an applet that runs either as
// an applet or a standalone application. To run
// standalone, it provides a main method that creates
// a frame, then creates an instance of the applet and
// adds it to the frame.

public class StandaloneApplet extends Applet
{
    public void init( )
    {
        add(new Button("Standalone Applet Button"));
    }

    public static void main(String args[ ])
    {
// Create the frame this applet will run in
        Frame appletFrame = new Frame("Some applet");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// Create an instance of the applet
        Applet myApplet = new StandaloneApplet();

// Initialize and start the applet
        myApplet.init();
        myApplet.start();

// The frame needs a layout manager
        appletFrame.setLayout(new FlowLayout());

// Add the applet to the frame
        appletFrame.add(myApplet);

// Have to give the frame a size before it is visible
        appletFrame.resize(300, 100);

// Make the frame appear on the screen
        appletFrame.show();

    }
}
```

21.7.4　为框架增加菜单

用户可以把 MenuBar 类联接到框架上，这样就可以为框架提供创建下拉式菜单的能力。用户可以使用下面的语句创建菜单：

```
MenuBar myMenuBar = new MenuBar( );
```

当用户创建完菜单条后，可以使用 setMenuBar 方法把它加到框架上：

```
myFrame.setMenuBar(myMenuBar);
```

有了菜单条以后，可以使用 add 方法在菜单条上增加菜单：

```
public synchronized Menu add(Memu newMenu)
```

下面的代码片段创建一个名为“File”的菜单并将该菜单加到菜单条上：

```
Menu fileMenu = new Menu("File");
myMenuBar.add(fileMenu);
```

某些窗口化操作系统允许用户创建当释放鼠标按钮时出现的菜单。这种菜单叫做撕开式菜单 tear-off menus。用户可以使用下面的语法在创建菜单时指定它为撕开式的菜单：

```
public Menu(String menuLabel, boolean allowTearoff)
```

除了增加子菜单以外，用户还可以在菜单中增加菜单项。菜单项是菜单的组成部分，通过菜单项用户可以实际地作出选择。从另一个角度说，菜单可用于包含菜单项和子菜单。例如，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在许多系统的 File 菜单中都包含诸如 New(新建文件)、Open(打开文件)、Save(存储文件)和 Save As(另存为)菜单项。如果用户创建了一个无菜单项的菜单结构,那么这种菜单结构是没有用途的。在这种菜单结构中没有东西可选择。用户可以使用两种方法为菜单添加菜单项。一种方法是简单地添加菜单项名称:

```
fileMenu.add("Open"); //在文件菜单中增加"Open"选项
```

用户也可在菜单中增加 MenuItem 类的实例:

```
MenuItem saveMenuItem = new MenuItem("save");//创建"save"菜单项
fileMenu.add(saveMenuItem); //把"save"选项加到文件菜单中
```

用户可以使用 enable 方法和 disable 方法激活或禁止菜单项。当用户禁止某个菜单项时,该菜单项仍然出现在菜单上,但是它一般都是灰色的(取决于窗口化的系统)。用户不能选择被禁止的菜单项。enable 方法和 disable 方法的格式是:

```
saveMenuItem.disable();//禁止文件菜单的存储选项
saveMenuItem.enable();//重新激活存储选项
```

除菜单项以外,用户还可以在菜单中增加子菜单和菜单分隔符。分隔符是出现在菜单中的一条线,它用来将菜单分成多个部分。要想添加分隔符,用户可以调用 addSeparator()方法:

```
public void addSeparator()
```

要想创建子菜单,用户只要创建一个新的菜单实例并将它加进当前的菜单即可:

```
Menu printSubmenu = new Menu("Print")
fileMenu.add(printfSubmenu);
printSubmenu.add("Print Preview");
                    //将打印预览作为打印菜单的选项加进菜单中
printSubmenu.add("print Document");
                    //将打印文档作为打印选项加进菜单中
```

用户也可以创建特殊的复选框菜单选项。这种选项的功能类似于复选框按钮。当用户第一次选择某个选项时,复选框中打上一个对号,表示该选项被选中。当用户再一次选择它时,复选框中的对号消失,表明该选项未被选中。为了创建复选框选项,可以使用下述方法:

```
public CheckboxMenuItem(String itemLabel)
```

如果复选框菜单项被选中,getState 方法的返回值为 true:

```
public void setState(boolean newState)
```

正常情况下,菜单是按照从左到右的顺序加进菜单条的。许多窗口化系统都在菜单条的较靠右边的位置上创建 Help 菜单。用户可以使用 setHelpMenu 方法在用户的菜单条设置这种菜单:

```
public synchronized void setHelpMenu(Menu helpMenu)
```

21.7.5 使用菜单

当菜单项被选中时,Java 或者产生一个动作事件,或者调用选项定义的动作方法,这取决

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

于用户使用的事件模式(Java 1.0 或 Java 1.1)。在 Java 1.0 中,action 方法的 whichAction 参数是被选中菜单项的名字:

```
public boolean action(Event evt,Object whichAction)
{
//首先,确保该事件是菜单选择事件
    if (evt.target instanceof MenuItem)
    {
    if ((String)whickAction = ="Save")
    {
            //处理存储选项事件
    }
  }
  return true;
}
```

在 Java 1.1 中,该事件模式已经改变了(该模式在 1.2 版本中得到了延用);在 1.1 事件模式下,用户必须为菜单创建 ActionListener。实现 ActionListener 的对象必须支持 actionPerfomed()方法,actionPerformed()方法用于接收动作发生的通知:

```
public void actionPerformed(ActionEvent event)
{
  if (event.getSource( ) instanceOf MenuComponet)
  {
    if (event.getSource( ) = = saveMencComponent)
    {
        //处理存储选项事件
    }
  }
}
```

程序清单 21.3 是一个应用程序的代码,它设置的菜单包含有 New, Open 和 Save 菜单项;另外复选框可以调用自动存储方法;打印子菜单还有两个选项。

```
import java.awt.*;
import java.applet.*;

public class MenuApplication extends Object
{
      public static void main(String[] args)
      {
// Create the frame and the menubar
          Frame myFrame = new Frame("Menu Example");
          MenuBar myMenuBar = new MenuBar();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// Add the menubar to the frame
        myFrame.setMenuBar(myMenuBar);

// Create the File menu and add it to the menubar
        Menu fileMenu = new Menu("File");
        myMenuBar.add(fileMenu);

// Add the New and Open menuitems
        fileMenu.add(new MenuItem("New"));
        fileMenu.add(new MenuItem("Open"));

// Create a disabled Save menuitem
        MenuItem saveMenuItem = new MenuItem("Save");
        fileMenu.add(saveMenuItem);
        saveMenuItem.disable();

// Add an Auto-Save checkbox, followed by a separator
        fileMenu.add(new CheckboxMenuItem("Auto-Save"));
        fileMenu.addSeparator();

// Create the Print submenu
        Menu printSubmenu = new Menu("Print");
        fileMenu.add(printSubmenu);
        printSubmenu.add("Print Preview");
        printSubmenu.add("Print Document");

// Must resize the frame before it can be shown
        myFrame.resize(300, 200);

// Make the frame appear on the screen
        myFrame.show();
    }
}
```

图 21.2 显示了 MenuApplication 程序的菜单，其中的打印文档选项被选中。

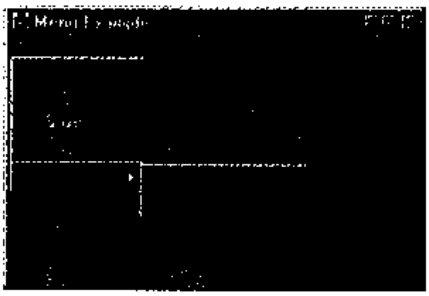


图 21.2 AWT 提供了许多流行的菜单特性，包括复选菜单项，禁止菜单项和分隔符

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

21.7.6　弹出式菜单

在编写程序时，人们经常希望为某个部件创建弹出式菜单，当用户用鼠标右键或中间的键单击该部件时产生该弹出式菜单。在 Java 1.1 中用户可以创建这种菜单。

创建弹出式菜单的做法与创建普通菜单的做法是相同的。用户首先需要实例化弹出式菜单，为此需要使用下列构造器：

```
public PopupMenu( )
public PopupMenu(String title)
```

然后用户在弹出式菜单中增加 MenuItem 对象，就像普通菜单一样。当用户加完所有选项之后，就可以使用部件的 add 方法将弹出式菜单加到部件上，如下面的代码片段所示：

```
PopupMenu popup = new PopupMenu("ButtonStuff");
popup.add("Winken")
popup.add("Blinken");
popup.add("Nodd");
Button myButton = new Button("Push Me")
myButton.add(popup);
```

21.8　对话框

对话框是弹出式窗口，它没有框架那么灵活。用户可以创建有模式对话框，也可以创建无模式对话框。术语“模式”意味着当对话框被显示时，禁止用户对其它窗口执行输入操作。当用户需要停止一切工作去获取对某个严重问题的回答时这种对话框是非常有用的，比如“Are you sure you want to quit?”。控制面板是一个无模式对话框的例子，控制面板可以在应用程序正在运行的过程中更改设置。

21.8.1　创建对话框

要想创建对话框，用户必须首先创建框架。对话框不能直接隶属于小应用程序。但是，小应用程序可以创建框架，而对话框可以隶属于框架。在创建对话框时，用户必须指明对话框是有模式的还是无模式的，并且一经建立完毕的对话框就再也不能修改它的“模式属性”了。

```
public Dialog(Frame parentFrame,boolean isModal)
```

下面的例子创建了一个有模式对话框，它的父容器是 myFrame：

```
Dialog myDialog = new Dialog(myFrame,true); //true 意味着是有模式对话框
```

用户也可以创建有标题的对话框：

```
public Dialog(Frame parentFrame,String title,boolean isModal)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:因为对话框不属于小应用程序,所以对话框在使用过程中可能会受到某些限制。解决的办法是创建一个哑框架来作为对话的容器。不幸的是这样用户就无法创建有模式对话框了,因为框架和框架上的部件都有它们自己的输入块,它们会继续它们的活动。比较好的解决方案是使用在本章前面的"框架"部分介绍的技术来实现。在这种情况下,用户应该使用框架创建一个独立运行的应用程序,用一个小的小应用程序段创建一个框架,然后在该框架中运行实际的小应用程序。

创建了对话框后,可使用 show 方法使它可见:

```
myDialog.show();
```

对话框类有几个与框架类共同的方法:

```
void setResizable(boolean);
boolean isResizable( );
void setTitle(String);
String getTitle( );
```

另外,如果对话框是有模式的,那么 isModal 方法将返回 true:

```
public boolean isModal()
```

21.8.2 可重用的 OK 对话框

程序清单 21.4 显示了 OKDialog 类,该类提供了一个 OK 对话框,该对话框显示一则信息并等待用户单击 OK 按钮。通常用户必须为对话框提供一个框架,但是当用户运行小应用程序时并没有创建框架,为了使小应用程序能够使用对话框,OKDialog 类提供了一个静态的 createOKDialog 方法用于为对话框创建框架。框架是被作为静态变量存储的,所以其它的对话框也可以使用同一个框架。

```
import java.awt.*;

//
// OKDialog - Custom dialog that presents a message and waits for
// you to click on the OK button.
//
// Example use:
//     Dialog ok = new OKDialog(parentFrame, "Click OK to continue");
//     ok.show();     // Other input will be blocked until OK is pressed
// As a shortcut, you can use the static createOKDialog that will
// create its own frame and activate itself:
//     OKDialog.createOKDialog("Click OK to continue");
//

public class OKDialog extends Dialog
{
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
    protected Button okButton;
    protected static Frame createdFrame;

    public OKDialog(Frame parent, String message)
    {
        super(parent, true);     // Must call the parent's constructor

// This Dialog box uses the GridBagLayout to provide a pretty good layout.

        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints constraints = new GridBagConstraints();

// Create the OK button and the message to display
        okButton = new Button("OK");
        Label messageLabel = new Label(message);

        setLayout(gridbag);

// The message should not fill, it should be centered within this area, with
// some extra padding. The gridwidth of REMAINDER means this is the only
// thing on its row, and the gridheight of RELATIVE means there should only
// be one thing below it.
        constraints.fill = GridBagConstraints.NONE;
        constraints.anchor = GridBagConstraints.CENTER;
        constraints.ipadx = 20;
        constraints.ipady = 20;
        constraints.weightx = 1.0;
        constraints.weighty = 1.0;
        constraints.gridwidth = GridBagConstraints.REMAINDER;
        constraints.gridheight = GridBagConstraints.RELATIVE;

        gridbag.setConstraints(messageLabel, constraints);
        add(messageLabel);

// The button has no padding, no weight, takes up minimal width, and
// Is the last thing in its column.

        constraints.ipadx = 0;
        constraints.ipady = 0;
        constraints.weightx = 0.0;
        constraints.weighty = 0.0;
        constraints.gridwidth = 1;
        constraints.gridheight = GridBagConstraints.REMAINDER;

        gridbag.setConstraints(okButton, constraints);
        add(okButton);

// Pack is a special window method that makes the window take up the minimum
// space necessary to contain its components.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        pack();
    }

// The action method just waits for the OK button to be clicked and
// when it is it hides the dialog, causing the show() method to return
// back to whoever activated this dialog.

    public boolean action(Event evt, Object whichAction)
    {
        if (evt.target == okButton)
        {
            hide();
            if (createdFrame != null)
            {
                createdFrame.hide();
            }
        }
        return true;
    }

// Shortcut to create a frame automatically, the frame is a static variable
// so all dialogs in an applet or application can use the same frame.

    public static void createOKDialog(String dialogString)
    {
// If the frame hasn't been created yet, create it
        if (createdFrame == null)
        {
            createdFrame = new Frame("Dialog");
        }
// Create the dialog now
        OKDialog okDialog = new OKDialog(createdFrame, dialogString);

// Shrink the frame to just fit the dialog
        createdFrame.resize(okDialog.size().width,
            okDialog.size().height);

// Show the dialog
        okDialog.show();

    }
}
```

程序清单 21.5 中的 DialogApplet 当用户按下鼠标按钮时创建一个 OK 对话框。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.awt.*;
import java.applet.*;

// DialogApplet
//
// Dialog applet creates a button, and when you press
// the button it brings up an OK dialog. The input
// to the original button should be blocked until
// the OK button in the dialog is pressed.

public class DialogApplet extends Applet
{
        protected Button launchButton;

        public void init()
        {
            launchButton = new Button("Give me an OK");
            add(launchButton);
        }

        public boolean action(Event event, Object whichAction)
        {
// Make sure this action is for the launchButton
            if (event.target != launchButton)
            {
                return false;
            }

// Create and display the OK dialog
            OKDialog.createOKDialog(
                "Press OK when you are ready");

// Signal that you've handled the event
return true;
        }
}
```

图 21.3 显示了弹出 OK 对话框的 DialogApplet。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

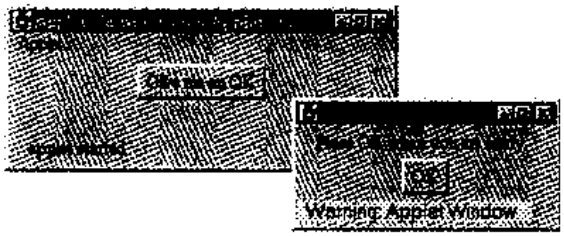


图 21.3 OKDialog 类创造了一个带 OK 按钮的弹出式对话框

21.9 滚动窗格

ScrollPane 是一个特殊的容器，该容器包含滚动条，滚动条允许用户滚动容器内部的内容。使用 ScrollPane 可以创建比较大的容器，容器可以超出屏幕的范围。ScrollPane 最常见的用途是显示大图像。用户可以创建一个显示图像的画布，然后将画布放置在 ScrollPane 容器里，由容器提供图像的自动翻滚功能。

用户可以通过滚动条控制滚动窗格。在缺省情况下，滚动窗格只在需要的时候才使用滚动条。用户可以指定一直使用滚动条(在这种情况下与 Panel 对象不同)。如果用户使用缺省的构造器，那么滚动窗格就在需要的时候使用滚动条，否则用户可以传递下列参数给构造器。这些参数包括 ScrollPane.SCROLLBARS _ ALWAYS、ScrollPane.SCROLL BARS _ NEVER 和 ScrollPane.SCROLLBARS _ AS _ NEEDED。下面是两个构造器：

```
public ScrollPane( )
public ScrollPane(int scrollbarOption)
```

用户可以使用给其它容器添加部件相同的方法为滚动窗格添加部件。用户可以通过调用 setScrollPosition 来设置可视区的位置，setScrollPosition 既可以用 point 对象为参数，也可以用 x，y 坐标值为参数。

```
public void setScrollPosition(Point point)
public void setScrollPosition(int x, int y)
```

setScrollPosition 方法只控制视区的左上角。其它的位置由滚动窗格的大小决定的。

如果用户希望监听来自于滚动窗格的滚动条事件，用户可以调用 getHAdjustable 方法和 getVAdujstable 方法来获得水平滚动条和垂直滚动条的 Adjustable 接口：

```
public Adjustable getHAdjustable( )
public Adjustable getVAdjustable( )
```

Adjustable 接口允许用户使用 setAdjustableListener 监听事件。

用户也可以使用 getViewport 方法确定视区的宽度和高度：

```
public Dimension getViewport( )
```

21.10 再谈布局管理器

不知用户是否已经注意到，当在容器中增加部件时，并没有告诉容器将部件放在什么位

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

置。通过使用布局管理器，可以告诉 AWT 用户希望如何相对其它部件来摆放新部件。布局管理器负责规划摆放它们的位置。这一特性有助于用户编写与平台无关的软件。当用户使用绝对坐标摆放部件时，如果用户把一个为 1280×1024 分辨率窗口设计的小应用程序拿到 640×480 分辨率的 Windows 95 上运行，将会出现部件聚堆的现象。

AWT 提供了五种不同类型的布局管理器：

- 流式布局(FlowLayout)　该布局方式将部件从左向右摆放，当一行摆满后换一行继续从左向右摆放。
- 网格布局(GridLayout)　该布局将容器分成尺寸相同的网格，部件被放置在网格空白处，顺序与流式布局一样，也是从上到下、从左向右摆放。网格布局与流式布局的不同在于网格布局给每个部件以相同的摆放空间。
- 边缘布局(BorderLayout)　边缘布局方式把容器看做是一个罗盘。当用户在容器中添加部件时，用户需要询问边缘布局在下面五个方位中的哪一个摆放部件："北"、"南"、"东"、"西"和"中"。它负责基于部件的相对尺寸规划出部件摆放的精确位置。
- 卡片式布局(CardLayout)　卡片式布局将加到容器里的部件看作是一个卡片堆栈，它把每个部件摆放在单独的卡上，在某一时刻只能有一个部件可见。
- 网格块布局(GridBagLayout)　网格块布局是最灵活的布局管理器。也是最容易搞混乱的布局管理器。网格块布局将容器作为单元网格看待，但与网格不同的是部件可以占用多个单元。当用户向网格块布局管理器管理的容器添加部件时，用户给出一个 GridBagConstraint，其中含有部件摆设位置和占用空间大小的信息。

21.10.1　流式布局

FlowLayout 类把容器看作一个行集。行高由摆放在行中的部件高度决定。FlowLayout 总是按照从左到右的顺序摆放新部件。如果在当前行不能摆放下一个部件，那么 FlowLayout 就到下一行重新按照从左到右的顺序摆放部件。它还试图按照左对齐、右对齐和居中方式对齐所有的行。FlowLayout 的缺省对齐方式是居中的，即当创建一行新部件时，它试图使部件的左右边缘留出相等的空隙。

> **提示**：FlowLayout 布局管理器是所有的小应用程序缺省的布局管理器。

FlowLayout 类的空构造器创建一个居中对齐的流式布局：

```
public FlowLayout()
```

用户也可以在创建流式布局时指定对齐方式：

```
public FlowLayout(int alignment)
```

FlowLayout 的对齐方式有 FlowLayout.LEET、FlowLayout.RIGHT 和 FlowLayout.CENTER。

用户也可以给出 FlowLayout 的水平空隙值和垂直空隙值。这些值指定了部件之间留出的最大空隙值和最小空隙值。这些空隙是以屏幕的像素点为单位给出的。

```
public FlowLayout(int alignment , int hgap , int vgap)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下面的代码片段创建了一个右对齐的 FlowLayout,其中水平间隙为 10 个像素,垂直间隙为 5 个像素:

```
myFlowLayout = new FlowLayout(FlowLayout.REGHT,10,5);
```

图 21.4 显示了 5 个按流式布局的按钮。

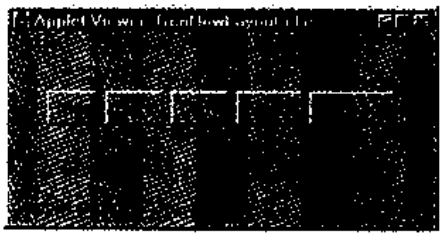


图 21.4 流式布局从左向右摆放部件

21.10.2 网格布局

GridLayout 类把容器分成单元尺寸相等的网格。当用户在容器中添加部件时,GridLayout 将它们从容器的左上角开始从左向右依次排列。当用户创建 GridLayout 类时,用户必须说明要使用的行数或列数。如果用户给出行数,那么布局管理器负责计算出列数。同样用户也可以给出列数,这时布局管理器负责计算出行数。如果用户为容器在两行内增加六个部件,布局管理器会自动地创建三个列。gridLayout 构造器的格式是:

```
public PridLayout(int numberOfRows , int numberOfColumns)
```

如果用户要创建一个行数固定的 GridLayout,那么应当使用 0 作为列数。如果用户要创建一个列数固定的 GridLayout,那么就应当使用 0 作为行数。

> **注意**:如果用户同时为 GridLayout 的行数和列数传递了非 0 值,布局管理器将只使用行数。而列数将根据部件数和行数重新计算。GridLayout(3,4)与 GridLayout(3,0)完全相同。

用户也可以指定水平间隙和垂直间隙:

```
public GridLayout(int rows ,int cols ,int hgap, int vgap)
```

下列的代码创建一个有四个列的 GridLayout,其中水平间隙是 8,垂直间隙是 10:

```
gridLayout myGridLayout = new GridLayout(0,4,8,10);
```

图 21.5 显示了按网格布局的 5 个按钮。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

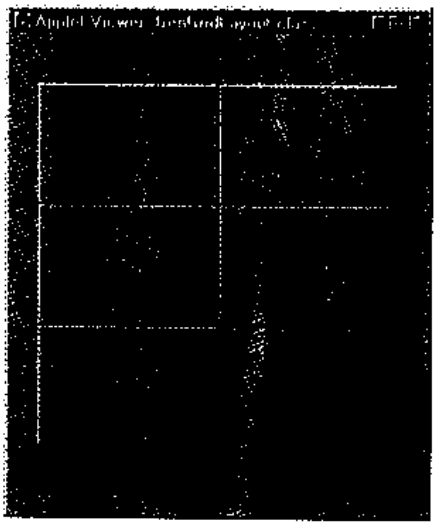


图 21.5　网格布局为每个部件分配相同大小的摆放区域

21.10.3　方位布局

BorderLayout 类把容器分成 5 个区域："北"、"南"、"东"、"西"和"中"。当用户向容器添加部件时，用户必须使用特殊的 add 方法的格式，必须包含这五个区域名中的一个。这五个区域的排列方式很像罗盘上这五个点。加到"北"区的部件放在容器的顶部，加在"西"区的部件放在容器的左边。

BorderLayout 类不允许将两个以上的部件放在同一区域内。用户可以选择地指定水平间隙和垂直间隙。要想创建不指定间隙的 BorderLayout，用户可以使用空构造器：

```
public BorderLayout( )
```

用户也可以指定水平间隙和垂直间隙：

```
public BorderLayout(int hgap,int vgap)
```

下面这行代码将 myButton 加在 BorderLayout 的"西区"：

```
myBorderLayout.add("West", myButton);
```

警告：

BorderLayout 类对增加部件的方法有限制，它与其他布局管理器不同。它的 add 方法使用附加的字符串作为参数。如果试图使用通常的 add 方法，则会得到一个错误信息。用户不能多次添加部件；如果在同一个区域添加了两个部件，用户将看到最后一个部件。

程序清单 21.6 显示了 BorderLayoutApplet 创建一个 BorderLayout 布局，并将它联接到当前的小应用程序中，然后在该小应用程序上增加几个按钮。

```
import java.applet.*;
import java.awt.*;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
//
// This applet creates a BorderLayout and attaches it
// to the applet. Then it creates buttons and places
// in all possible areas of the layout.

public class BorderLayoutApplet extends Applet
{
    public void init()
    {

// First create the layout and attach it to the applet

        setLayout(new BorderLayout());

// Now create some buttons and lay them out

        add("North", new Button("Larry"));
        add("South", new Button("Curly Joe"));
        add("East", new Button("Curly"));
        add("West", new Button("Shemp"));
        add("Center", new Button("Moe"));
    }
}
```

图 21.6 显示了按照方位布局的 5 个按钮。

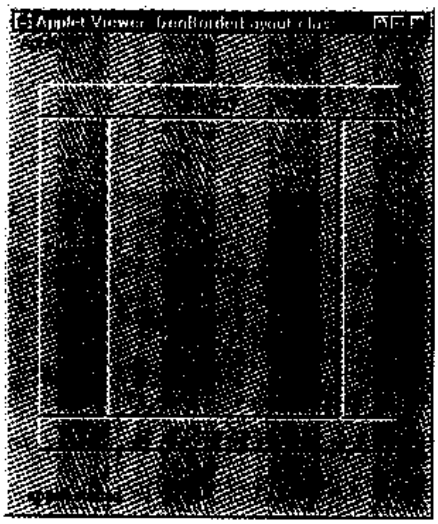


图 21.6 方位布局将部件放置在罗盘点“北”、“南”、“东”、“西”和“中”上

21.10.4 网格块布局

GridBagLayout 类像 GridLayout 类一样把容器分成尺寸相等的网格单元。但与 GridLay-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

out 不同是 GridBagLayout 类决定行数和列数并允许一个部件在必要时占用多个网格单元。部件占用的所有单元叫做显示区域。用户在容器内添加部件之前，必须给 GridBagLayout 一个在何处摆放部件的“建议”集。这些建议是以 GridBagConstraints 类的形式给出的。GridBagConstraints 类有许多控制部件位置的变量：

- gridx 和 gridy：gridx 和 gridy 单元的坐标，下一个部件将放置在该坐标上（如果部件占用多个单元，该坐标将是部件左上角占用的单元的坐标）。GridBagLayout 的左上角坐标是 0，0。gridx 和 gridy 的缺省值是 GridBagConstraints.RELATIVE，对于 gridx 这个值意味着位于最后被加进来的部件的右方；对于 gridy 这个值意味着最后被加进来的部件的下方。

- gridwidth 和 gridheight：gridwidth 和 gridheight 代表部件占用的单元宽度和单元高度。gridwidth 和 gridheight 缺省值为 1。如果用户希望这个部件是某一行的最后一个，可以使用 GridBagConstraint.REMAINDER 作为 gridwidth 的值（如果该部件是某列的最后一个，那应将作为 gridheight 的值）。如果该部件是某行或某列的倒数第二个部件，那么应该使用它的 GridBagConstraint.RELATIVE 值。

- fill：fill 告诉 GridBagLayout 当部件小于显示区域时该干什么。fill 的缺省值是 GridBagConstraint.NONE，该值指示部件的尺寸保持不变。GridBagConstraint.HORIZONTAL 值指示部件沿水平方向伸展全部占满水平显示区，而垂直方向不变。GridBagConstraint.VERTICAL 指示部件沿垂直方向伸展而水平方向不变。GridBagConstraint.BOTH 指示部件沿垂直和水平两个方向伸展并全部占满所有的显示区域。

- ipadx 和 ipady：ipadx 和 ipady 告诉 GridBagLayout 在部件的 x 方向和 y 方向增加多少个像素。这些像素将被加在部件的两边，所以 ipadx 为 4 将引起部件的左右两边各增加四个像素。用户应当牢记部件的尺寸将增加两倍的填补量，因为填补是发生在两边的。ipadx 和 ipady 缺省值是 0。

- insets：insets 变量是 Insets 类的一个实例，它指示部件边缘和显示区边缘留多少空隙。换句话说，insets 在部件的周围创建一块空旷的“无人地带”。Insets 类（将在本章后面的“镶边”部分中讨论）有单独的顶边值，底边值、左边值和右边值。

- anchor：当部件小于显示区时使用 anchor。它指示部件在显示区中摆放的位置。缺省值是 GridBagConstraint.CENTER，该值指示部件摆放在显示区的中心。其它的值全部是罗盘点：

```
GridbagConstraints.NORTH
GridBagConstraints.NORTHEAST
GridBagConstraints.EAST
GridBagConstraints.SOUTHEAST
GridBagConstraints.SOUTH
GridBagConstraints.SOUTHWEST
GridBagConstraints.WEST
GridBagConstraints.NORTHWEST
```

和 BorderLayout 类一样，NORTH 表示屏幕顶部，EAST 表示右边。

- weightx 和 weighty：这两个变量用于设置部件的相对尺寸。例如，具有 weightx 值2.0 的部件占据水平空间两倍于具有 weightx 值 1.0 的部件占据的水平空间。因为这些值是相对的，所以对于在同一行的有着 1.0 或 3.0 的权值的所有部件没有什么差别。用户至少要为一

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

个部件指明水平和垂直两个方向的权值，否则 GridBagLayout 将向容器中间挤压这些部件。

当用户想要使用 GridBagLayout 类在容器中增加部件时，用户应当首先创建部件，然后再创建 GridBagCoustraints 的实例并为该部件设定限制。例如：

```
GridBagLayout myGridBagLayout = new GridBagLayout ( );
setLayout(myGridBagLayout);
        //将小应用程序的布局管理器设置成网格布局
Button myButton = new Button("My Button");
GridBagConstraints constraints = new GridBagConstraints ( );
constraints.weightx = 1.0;
constraints.gridwidth = GridBagConstraints.RELATIVE;
constraints.fill = GridBagConstraints.BOTH;
```

然后，用户再在 GridBagLayout 中为部件设置限制：

```
myGridLayout.setCoustraints(myButton,constraints);
```

现在用户就可以把部件加到容器中了：

```
add(myButton)
```

程序清单 21.7 中的小应用程序使用了 GridBagLayout 类安排几个 CircleCanvas 类的实例（在第 19 章的“画布”部分中创建的 circleCanvases 类）。

```
import java.applet.*;
import java.awt.*;

//
// This circle demonstrates the CircleCanvas class we
// created. It also shows you how to use the GridBagLayout
// to arrange the circles.

public class CircleApplet extends Applet
{
    public void init()
    {
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints constraints = new GridBagConstraints();
        CircleCanvas newCircle;

        setLayout(gridbag);

// You'll use the weighting to determine relative circle sizes. Make the
// first one just have a weight of 1. Also, set fill for both directions
// so it will make the circles as big as possible.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        constraints.weightx = 1.0;
        constraints.weighty = 1.0;
        constraints.fill = GridBagConstraints.BOTH;

// Create a red circle and add it

        newCircle = new CircleCanvas(Color.red);
        gridbag.setConstraints(newCircle, constraints);
        add(newCircle);

// Now, you want to make the next circle twice as big as the previous
// one, so give it twice the weight.

        constraints.weightx = 2.0;
        constraints.weighty = 2.0;

// Create a blue circle and add it

        newCircle = new CircleCanvas(Color.blue);
        gridbag.setConstraints(newCircle, constraints);
        add(newCircle);

// You'll make the third circle the same size as the first one, so set the
// weight back down to 1.

        constraints.weightx = 1.0;
        constraints.weighty = 1.0;

// Create a green circle and add it.

        newCircle = new CircleCanvas(Color.green);
        gridbag.setConstraints(newCircle, constraints);
        add(newCircle);

    }
}
```

图 21.7 显示了 GridBagApplet 创建的三个圆形画布。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

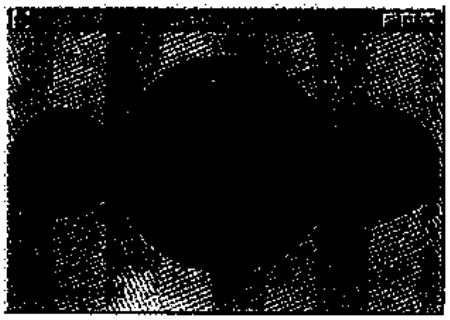


图 21.7 GridBagApplet 创建的三个圆形画布

21.11 镶边

镶边不是布局管理器。它们是指示布局管理器在容器的边缘留出多大空隙的指令。换句话说,镶边定义了容器的边缘与容器包含的部件之间留出的空白区域。如果用户在容器的左边有一个 20 个像素的镶边,那么在距容器左边缘 20 个像素范围内不会摆放任何部件。

镶边是由 Insets 类的实例描述的。这个类有几个实例变量;它们是左镶边值、右镶边值、顶镶边值和底镶边值。布局管理器通过调用容器的 insets 方法确定每个容器的镶边值,insets 方法返回一个 Insets 类的实例。例如,如果用户想要为小应用程序镶一个 20 个像素的边并为小应用程序中的部件之间留出 20 个像素的间隙,那么用户就应该在小应用程序中创建一个 insets 方法:

```
public Insets insets( )
{
  return new Insets (20,20,20,20);
  //在容器四周镶一个 20 个像素宽的边
}
```

这时 Insets 类的构造器获得四个镶边值,其顺序是顶边、左边、底边和右边。

图 21.8 显示的结果好像是 GridBagApplet 使用了上面的 insets 方法。圆之间的间隙不是来自于 Insets 类,而是来自于圆被缩小。顶部、底部、左边和右边的空隙是由 Insets 类创建的。

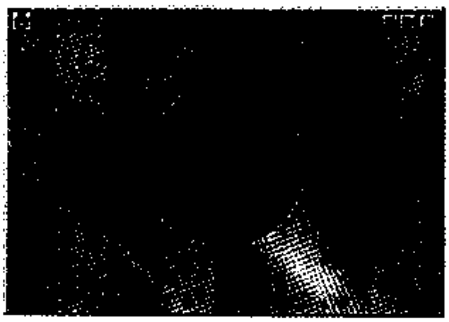


图 21.8 Insets 创建了部件与容器边缘间的空隙

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

21.12　空布局管理器

虽然 Java 建议用户使用布局管理器，但并不要求用户必须使用布局管理器。在有些情况下用户需要在一定的坐标上显式地放置部件。如果用户在 null(空)容器中设置了布局管理器，那么用户就可以使用每个部件的 move 方法和 resize 方法显式地设置部件的位置和尺寸。

21.13　Sun 公司的产品进一步扩展

Sun 公司正在开发更完善的 Java 应用程序开发环境，这个未来的开发环境名为 Solstice Workshop，它包含了一个健壮的类库集。这些库中将包含名为 Admin View Module(AVM)的类集，该类集将公布某些 AWT 的缺点。它完善了 AWT，但不取代 AWT。AVM 将包括下列有用的特性：

- 图形按钮
- 系列列表框
- 滚动窗口
- 工具条
- 图像画布
- 许多公用的对话框

这些特性将完整地集成在 Solstice Workshop 中，Solstice Workshop 允许用户不必编写太多代码就可以很快地开发健壮的应用程序。要想了解更多关于 AVM 的信息用户可以访问 Java 管理 API 网站 http://www.javasoft.com/products/ JavaManagement/index.html。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第22章 图　　形

本章内容：

22.1 Java 图形
22.2 paint 方法、Update 方法和 repaint 方法
22.3 Graphics 类
22.4 Polygon 类
22.5 画文本
22.6 画模式
22.7 绘制图像
22.8 MediaTracker 类
22.9 实用图形类
22.10 Color 类
22.11 裁剪
22.12 动画技术
22.13 打印

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

22.1　Java 图形

抽象窗口化工具(AWT)为公共用户界面组件,比如按钮和菜单提供了一个应用编程接口(API)。

Java 的主要目标之一是要提供一个与平台无关的开发环境。图形用户界面(GUI)总是最难以移植的一部分。Windows 的 API 与 OS/2 Presentation Manager API 有区别,OS/2 Presentation Manager 的 API 与 X-Windows API 有区别,X-Windows API 与 Mac API 也有区别。这个问题最普通的解决办法是考察所有要使用的平台,找出各种平台共用的组件(在各种平台上很容易实现的),然后创建一个单独的可以在各种平台上使用的 API。在各种不同的平台上,该公用的 API 是应用程序与该平台上的本地 API 的接口,使用公用的 API 的应用程序看上去会感觉与使用本地的 API 的应用程序具有相同的效果。

与此相对的解决办法是创建一个单独的外观和用法,然后再在各种平台上实现。对于 Java,Sun 公司选择了公用 API 解决方案 该方案使得 Java 应用程序可以较好地适应各种不同的平台,Sun 公司称这个公用的 API 为 Abstract Windowing Toolkit(抽象窗口化工具),简称 AWT。

抽象窗口化工具把图形分为两个不同的层次。在低层次上它处理原始图形功能和不同的输入设备,比如鼠标和键盘。在高层次上它提供了大量的组件,比如按钮和滚动条。

本章主要讨论低层次的图形和抽象窗口化工具的打印特性。第 27 章将讨论低层次的输入问题,第 28 章和第 29 章将讨论抽象窗口化工具的高层部分。

22.2　paint 方法、Update 方法和 repaint 方法

正如用户在简单的 HelloWorld 小应用程序中看到的那样,Java 小应用程序通过调用 paint 方法重画自己。因为小应用程序从不显式地调用 paint 方法,所以用户可能并不知道如何调用它。用户的小应用程序实际上有三种重画自己的不同方法,它们是:

- repaint 方法　当小应用程序需要被重画时调用 repaint 方法
- Update 方法　向 repaint 发出刷新小应用程序的信号。缺省的 Update 方法清除小应用程序的画图区并调用 paint 方法。
- paint 方法　在画图区中由 paint 方法实际地画出小应用程序的图形。将 Graphics 类的实例传递数,给 paint 方法,以便画出各种形状和图像。

22.3　Graphics 类

Graphics 类提供了画各种图形的方法,其中包括下列图形:

- 线
- 圆和椭圆
- 矩形和多边形
- 图像
- 各种字体的文本

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

另外 Graphics2D 类和 Graphics3D 类扩展了 Graphics 类。要想了解新的 2D 特性可以参考第 26 章“Java 二维图形”部分。

22.3.1 坐标系统

在 Java 中使用的坐标是简单的笛卡儿(x,y)坐标系统,其中 x 是从屏幕左边开始到当前点之间的水平像素数。y 是从屏幕的顶到当前点间的垂直像素数。屏幕的左上角用(0,0)表示。这几乎是在各种图形系统中通用的坐标系统。图 22.1 给出一些关于坐标的示例。

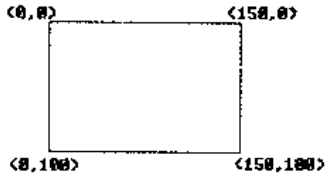


图 22.1 与数学中的 y 坐标自底向上增长不同,Java 中的 y 坐标是由顶向下增长的

22.3.2 画线

使用 Graphics 类可以画的最简单的图形是直线。drawLine 方法使用两对坐标即 x1, y1 和 x2,y2 作为参数并画出这两点间的直线:

```
public abstract void drawLine(int x1,int y1,int x2,int y2)
```

程序清单 22.1 中的小应用程序使用 drawLine 方法画了两条线。该小应用程序的输出如图 22.2 所示。

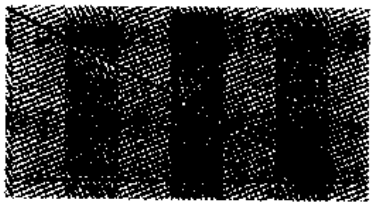


图 22.2 画线是最基本的图形操作

程序清单 22.1 DrawLines.java 的源代码

```
import java.awt.*;
import java.applet.*;

//
// This applet draws a pair of lines using the Graphics class
//

public class DrawLines extends Applet
{
  public void paint(Graphics g)
  {
// Draw a line from the upper-left corner to the point at (200, 100)
    g.drawLine(0, 0, 200, 100);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// Draw a horizontal line from (20, 120) to (250, 120)
    g.drawLine(20, 120, 250, 120);
  }
}
```

22.3.3　画矩形

既然用户已经知道了如何画线，那么用户就可以继续了解如何画矩形和填充矩形。要想画矩形，用户需要使用 drawRect 方法，同时应当向该方法传递矩形左上角的 x 和 y 坐标以及矩形的宽和高：

```
public abstract void drawRect(int x, int y, int width, int height)
```

要想在(150,100)处画一个 200 个像素宽 120 个像素点高的矩形，用户应该使用下面的方法：

```
g.drawRect(150,100,200,120);
```

drawRect 方法只画矩形的轮廓。如果用户想要画一个实心的框，用户可以使用 fillRect 方法，fillRect 方法使用与 drawRect 方法相同的参数：

```
public abstract void fillRect(int x , int y , int width,int height)
```

用户也可以使用 clearRect 方法清除一个区域，该方法也使用与 drawRect 完全相同的参数：

```
public abstract void clearRect(int x , int y , int width,int height)
```

图 22.3 显示了 drawRect 方法，fillRect 方法和 clearRect 方法之间的差别。其中左边的矩形是用 drawRect 方法画出来的，中间的实心矩形是使用 fillRect 方法画出来的。而右边的图形是在用 fillRect 方法画出的填充矩形之上使用 clearRect 方法清除了中间部分的图形。



图 22.3　Java 提供了几个可以灵活地画出矩形的方法

22.3.4　画三维矩形

Graphics 类还提供了一个画三维矩形(类似于工具条上的按钮)的方法。遗憾的是 Graphics 类画出的这些按钮只有很浅的深度或高度，因此这些三维效果不易看出。draw3DRect 方法和 fill3DRect 方法的语法与 drawRect 方法和 fillRect 方法的语法非常相似，唯一的一点差别是 draw3DRect 和 fill3DRect 多一个布尔型的参数，该参数指明矩形是否需要凸起：

```
public void draw3dRect (int x, int y, int width, int height, boolean raised)
public void fill3dRect (int x, int y, int width, int height, boolean raised)
```

凸起和下凹效果是通过在矩形周围画更亮或更暗的线条产生的。

假设光线来自于屏幕的左上角。那么凸起的三维矩形的迎光面是顶部和左边，而底部和右边产生阴影。下凹的三维矩形恰恰相反，它在顶部和左边产生阴影，而在底部和右边是迎光

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

光面。draw3DRect 和 fill3DRect 方法都使用亮色画凸起矩形的顶部和左边，用暗色画凸起矩形的的底部和右边。它们都使用暗色画下凹矩形的顶部和左边，用亮色画下凹矩形的底部和右边。另外，fill3DRect 方法还使用较暗的阴影画整个下凹按钮。程序清单 22.2 中的小应用程序画了一些凸起的或下凹的填充的或不填充的矩形。

```
import java.awt.*;*
import java.applet.*;

//
// This applet draws four varieties of 3-D rectangles.
// It sets the drawing color to the same color as the
// background because this shows up well in HotJava and
// Netscape.

public class Rect3d extends Applet
{
    public void paint(Graphics g)
    {
// Make the drawing color the same as the background
        g.setColor(getBackground());

// Draw a raised 3-D rectangle in the upper-left
        g.draw3dRect(10, 10, 60, 40, true);
// Draw a lowered 3-D rectangle in the upper-right
        g.draw3dRect(100, 10, 60, 40, false);

// Fill a raised 3-D rectangle in the lower-left
        g.fill3dRect(10, 80, 60, 40, true);
// Fill a lowered 3-D rectangle in the lower-right
        g.fill3dRect(100, 80, 60, 40, false);
    }
}
```

图 22.4 显示了 Rect3d 小应用程序的输出。对于凸起的矩形填充与不填充效果是一样的。这只是因为使用的颜色与背景的颜色是一致的。如果使用与背景不同的颜色，则填充的按钮将使用选定的颜色画出，而不填充的按钮将仍然显示背景颜色。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

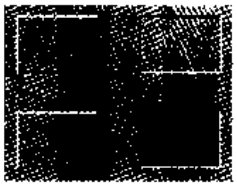


图 22.4　draw3DRect 方法和 fill3DRect 方法使用阴影来产生三维效果

22.3.5　画圆角矩形

除了规则矩形和三维矩形以外，用户还可以画带圆角的矩形。除了增加两个参数以外，drawRoundRect 方法和 fillRoundRect 方法与 drawRect 方法和 fillRect 方法是一样的：

```
public abstract void drawRoundRect(int x, int y, int width, int height,int arcWidth, int arcHeight)
public abstract void fillRound Rect(int x, int y, int width, int height, int arcWidth, int arcHeight)
```

arcWidth 参数和 arcHeight 参数指示圆角的大小。例如，arcWidth 为 10 告诉 Graphics 类在矩形的最左边和最右边各圆出一个 5 个像素宽的角。arcHeight 为 8 告诉 Graphics 类在矩形的最顶部和最底部各圆出一个 4 个像素高的角。

图 22.5 显示了圆角矩形的左上角。图中的 arcWidth 的值为 30，arcHeigt 的值 10。该图还显示了一个想象中的宽为 30 高为 29 的椭圆，用来辅助阐明如何圆角。

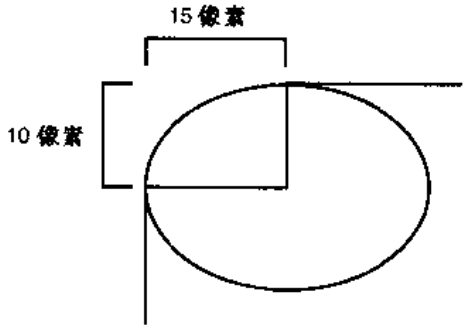


图 22.5　Java 使用椭圆来确定圆角的大小

程序清单 22.3 中的小应用程序画出了一个圆角矩形和一个填充的圆角矩形。图 22.6 显示了这个小应用程序的输出。



图 22.6　Java 可以很容易地使用圆角矩形替换尖角矩形

```
import java.awt.*;
import java.applet.*;

// Example 22.3 - RoundRect Applet
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
//
// This applet draws a rounded rectangle and then a
// filled, rounded rectangle.

public class RoundRect extends Applet
{
    public void paint(Graphics g)
    {
// Draw a rounded rectangle with an arcWidth of 20, and an arcHeight of 20
        g.drawRoundRect(10, 10, 40, 50, 20, 20);

// Fill a rounded rectangle with an arcWidth of 10, and an arcHeight of 8
        g.fillRoundRect(10, 80, 40, 50, 10, 6);
    }
}
```

22.3.6 画圆和椭圆

如果用户厌烦了方形形状,那么用户可以试着使用圆形。Graphics 类不区分圆和椭圆,所以 Graphics 类并没有 drawCircle 方法。它只有 drawOval 方法和 fillOval 方法:

```
public abstract void drawOval (int x, int y ,int width , int height)
public abstract void fillOval (int x, int y ,int width , int height)
```

要想画出圆和椭圆,用户首先要想到圆和椭圆的外接矩形。用户可以将该矩形的左上角坐标传递给 drawOval。同时用户还需要向 drawOval 方法传递椭圆宽度和高度。如果椭圆的高度和宽度相等,那么用户画出来的就是一个圆。图 22.7 阐明了外接矩形的概念。

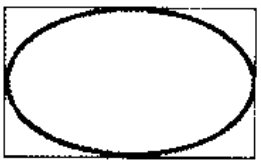


图 22.7 圆和椭圆都是画在想象中的外接矩形中的

程序清单 22.4 中的小应用程序画出了一个圆和一个填充椭圆。图 22.8 显示了该小应用程序的输出。



图 22.8 Java 不区分椭圆和圆,椭圆和圆都是被当作椭圆来处理的

```
import java.awt.*;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.applet.*;

//
// This applet draws an unfilled circle and a filled ellipse

public class Ovals extends Applet
{
    public void paint(Graphics g)
    {

// Draw a circle with a diameter of 30 (width = 30, height = 30)
// With the enclosing rectangle's upper - left corner at (0, 0)
      g.drawOval(0, 0, 30, 30);

// Fill an ellipse with a width of 40 and a height of 20
// The upper - left corner of the enclosing rectangle is at (0, 60)
      g.fillOval(0, 60, 40, 20);
    }
}
```

22.3.7　画多边形

用户也可以使用 Graphics 类画多边形和填充多边形。用户画多边形可以有两种选择。用户既可以传递两个包含多边形顶点的 x 和 y 坐标的数组，也可以传递 Polygon 类的实例：

```
public abstract void drawPolygon(int[] xPoints, int [] yPoints, int numPoints)
public void drawPolygon(Polygon p)
```

程序清单 22.5 中的小应用程序使用多点数组画出了一个多边形。图 22.9 显示了这个小应用程序的输出。



图 22.9　Java 允许用户画出可以想象的任意多边形

```
import java.applet.*;
import java.awt.*;

//
// This applet draws a polygon using an array of points

public class DrawPoly extends Applet
{
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// Define an array of X coordinates for the polygon
    int xCoords[] = { 10, 40, 60, 30, 10 };

// Define an array of Y coordinates for the polygon
    int yCoords[] = { 20, 0, 10, 60, 40 };

    public void paint(Graphics g)
    {
      g.drawPolygon(xCoords, yCoords, 5); // 5 points in polygon
    }
}
```

警告:注意在本例中的多边形不是封闭的。换句话说,在多边形的第一点和最后一点之间没有连线。如果用户想画封闭多边形,用户必须在数组的末尾重复第一点。

22.4 Polygon 类

Polygon 类提供了更加灵活的定义多边形的方法。用户可以通过传递顶点的 x 坐标数组和顶点的 y 坐标数组来创建多边形:

```
public Polygon(int[ ] xPoints ,int [ ] yPoints ,int numPoints)
```

用户也可以先创建一个空多边形,然后再逐次地在其中加点:

```
public Polygon( )
public void addPoint (int x,int y)
```

一旦用户创建了 Polygon 类的实例,用户就可以使用 getBounds 方法去确定该多边形占用的区域(x 坐标和 y 坐标的最小值和最大值):

```
public Rectangle getBounds()
```

由 getBounds 方法返回的 Rectangle 类包含有代表该矩形左上角 x 和 y 坐标的变量和代表矩形的宽度和高度的变量。用户也可以调用以某点的 x 坐标和 y 坐标为参数的 contains 方法来确定该点是否包含在该多边形范围内:

```
public boolean contains(int x,int y)
```

例如,用户可以使用下面的代码片段检查点(5,10)是否包含在 myPolygon 中:

```
if (myPolygon.contains(5,10))
{
    //点(5,10)在该多边形内部
}
```

用户可以使用 Polygon 类代替 drawPolygon 方法和 fillPolygon 方法中的点数组。程序清单22.6中的小应用程序创建了一个多边形类的实例并用它画出了一个填充多边形。图22.10

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

显示了这个小应用程序的输出。



图 22.10　使用 Polygon 类创建的多边形与使用点数组创建的多边形完全一样

程序清单 22.4　Polygons.java 的源代码

```
import java.applet.*;
import java.awt.*;

//
// This applet creates an instance of a Polygon class and then
// uses fillPoly to draw the Polygon as a filled polygon.

public class Polygons extends Applet
{
// Define an array of X coordinates for the polygon
    int xCoords[] = { 10, 40, 60, 30, 10 };

// Define an array of Y coordinates for the polygon
    int yCoords[] = { 20, 0, 10, 60, 40 };

    public void paint(Graphics g)
    {
// Create a new instance of a polygon with 5 points
        Polygon drawingPoly = new Polygon(xCoords, yCoords, 5);

// Draw a filled polygon
        g.fillPolygon(drawingPoly);
    }
}
```

22.5　画文本

Graphics 类还包含了画文本字符和文本字符串的方法。正如在小应用程序 Hello Word 中看到的那样，用户可以使用 drawString 方法在屏幕上画文本字符串。在介绍画文本之前，让我们熟悉一下与文本和字体有关的术语，如下所示：

- 基线　想象中排放文本的基准线。
- 底线　某些字符在基线下面伸出的部分。比如 j 字符在基线下面就伸出来一部分。
- 顶线　从基线算起字符占用的高度。比如 d 字符就比 x 字符的顶线高。
- 行距　某一行的底线到下一行的顶线之间的距离。如果没有行距，则上一行的 g 字符和 j 字符就与下一行的 M 字符和 H 字符接触上了。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

警告: Java 里的顶线术语与出版界的顶线术语的含意有些不同。出版界的顶线术语的含意是 x 字符的顶和某个字符之间的距离,而 Java 里的顶线术语指的是从基线到某个字符的顶这段距离。

图 22.11 阐述了顶线、底线、基线和行距之间的相互关系。

注意:用户可能也听说过术语均衡字体和固定字体。在固定字体中每个字符占用相同的空间。老式打字机(如果用户还记得)就是按固定字体打印的。均衡字体中的字符根据需要占用空间。下面我们以本书中使用的字体为例说明这两种字体。

本书中的正文使用的是均衡字体,这样可以使得读者看上去美观一些。用户注意一下这些字体就会发现其中的每个字符是根据需要占用的空间(例如,用户可以比较一下字符 I 和字符 m 所占空间)。本书中的代码例子都是使用固定字体书写的(保留了原来的空隙),所以每个字符都占用相等的空间。

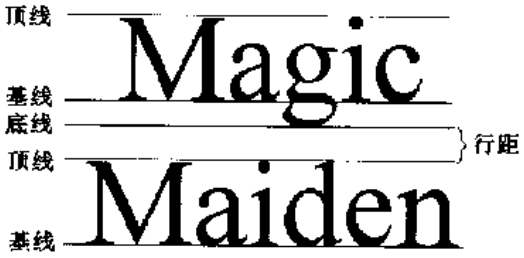


图 22.11 Java 的字体术语起源于排版领域,但是某些含意有些改变

要想使用 Graphics 类画字符串,用户可以调用 drawString 方法,同时要给出用户要画的字符串和基线开头的 x 和 y 坐标(这就是需要介绍术语的原因):

```
public abstract void drawString (String str , int x , int y)
```

用户可以回想一下"Hello World"小应用程序,它就是用这种方法画出那条著名信息的:

```
public void paint (Graphics g)
{
    g.drawString("Hello World", 10,30);
}
```

用户也可以画来自于字符串数组和字节数组的字符。drawChars 方法 drawBytes 方法的格式是:

```
void drawChars(char charArray[], int offset, int numChars, int x, int y)
void drawBytes(byte charArray[], int offset, int numChars, int x, int y)
```

offset 参数引用的是第一个要画的字符或字节的位置。通常它的值为零,因为用户一般都希望从数组的开始画起。程序清单 22.7 中的小应用程序画了一些来自字符数组的字符和来自字节数组的字符。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

程序清单 22.7　DrawChars.java 的源代码

```
import java.awt.*;
import java.applet.*;

//
// This applet draws a character array and a byte array

public class DrawChars extends Applet
{
    char[] charsToDraw = { 'H', 'i', ' ', 'T', 'h', 'e', 'r', 'e', '!' };

    byte[] bytesToDraw = { 65, 66, 67, 68, 69, 70, 71 }; // "ABCDEFG"

    public void paint(Graphics g)
    {
      g.drawChars(charsToDraw, 0, charsToDraw.length, 10, 20);

      g.drawBytes(bytesToDraw, 0, bytesToDraw.length, 10, 50);
    }
}
```

22.5.1　Font 类

用户可能发现用户的小应用程序中缺省的字体不是非常有趣。有幸地是，用户可以选择大量不同的字体。这些字体对于不同的系统可能有些差异，在实际应用中可能会产生移植问题，尤其是在将来。但是在现在，HotJava 和 Netescape 支持的是同一个字体集。

除了选择字体外，用户还可以选择字体风格：Font.PLAIN。Font.BOLD 和 Font.ITALIC。这些风格可以组合使用，所以用户可以通过设定 Font.BOLD + Font.ITALIC 而得到粗的斜体字体。

当选择字体时，用户还必须给出字体磅值。磅值是打印术语，指的是字体的大小。当在打印机上打印时，每英寸有 100 个点，但这并不一定适用于屏幕字体。Microsoft Windows 定义的磅值在各种分辨率的屏幕上具有相同的高度。换句话说同一台监视器在分辨率为 640 × 480 的屏幕上显示的 14 点字符具有的高度与在分辨率为 1024 × 768 的屏幕上显示的 14 点字符具有的高度是相同的。在 Java 中字体的高度直接随像素数变化。在 1280 × 960 分辨率下的 14 点字体的高度是在 640 × 480 模式下 14 点字体的高度的两倍。在 Java 中如此使用磅值是因为许多小应用程序使用了绝对屏幕坐标，尤其是在画原始图形时。直线和正方形是有固定高度的。如果用户使用这种外形画文本时，应该也使用固定高度。

用户可以使用字体名、字体风格和磅值创建字体实例：

```
public Font(String fontName , int style , int size)
```

下面的声明创建具有粗斜体风格 12 磅的 TimesRoman 字体

```
Font myFont = new Font ("TimesRoman",Font.BOLD + Font. ITALIC,12);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用户也可以使用 getFont 方法获取在系统属性中描述的字体：

```
public static Font getFont (String propertyName)
```

该方法返回由名为 propertyName 的系统属性描述的 Font 的实例。如该属性名未被设置，则该方法返回 null。

```
public static Font getFont(String propertyName , Font defaultValue)
```

该方法返回由名为 propertyName 的系统属性描述的 Font 的实例。如果该属性名未被设置，则该方法返回 defaultValue。

getFont 方法允许在系统属性中描述的字体拥有与字体名关联的风格和磅值。描述属性中的字体的格式是

```
font-style-pointsize
```

该 style 参数可以是 bold、italic、bolditalic，或者什么都不出现。如果 style 参数不出现，那么该字符串的格式为

```
font-pointsize
```

用户可以将系统属性中粗的 16 点的 TimesRoman 字体描述为：

```
TimesRoman-bold-16
```

这个机制用于设置特殊的字体种类。例如，用户可以编写个 Java VT－100 终端访真器程序，该程序使用系统属性 defaultVT100Font 查找显示文本的字体。用户可以在命令行中设置这样一个属性：

```
java -DdefaultVT100Font = courier-14 emulators. vt100
```

用户可以使用下面的方法获取关于字体的信息：

```
public String getFamily( )
```

字体族是一个与平台有关的字体名。它经常与字体名相同。

```
public String getName ( )
public int getSize( )
public int getstyle( )
```

用户可以通过分别检查粗体、斜体和无格式来检查字体的风格：

```
public boolean isBold( )
public boolean isItalic( )
public boolean isPlain( )
```

Toolkit 类的 getFontList 方法返回一个包含可用字体名字的字符串：

```
public abstract String[ ] getFontList( )
```

用户也可以使用 Toolkit 类的 getDefaultToolkit 方法获得对当前工具的引用：

```
public static synchronized Toolkit getDefaultToolkit( )
```

程序清单 22.8 中的小应用程序使用 getFontList 方法显示了各种风格的可用字体。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

图 22.12 显示了程序清单 22.8 的结果。

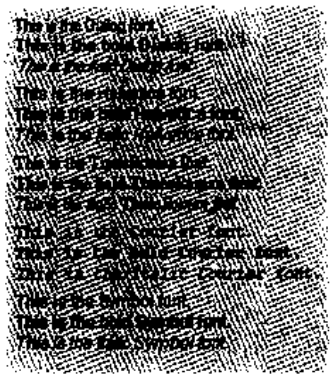


图 22.12　Java 提供了许多不同的字体和字体风格

程序清单 22.8　ShowFonts.java 的源代码

```
import java.awt.*;
import java.applet.*;

//
// This applet uses the Toolkit class to get a list
// of available fonts, then displays each font in
// PLAIN, BOLD, and ITALIC style.

public class ShowFonts extends Applet
{
    public void paint(Graphics g)
    {
      String fontList[];
      int i;
      int startY;

// Get a list of all available fonts
      fontList = getToolkit().getFontList();

      startY = 15;

      for (i=0; i < fontList.length; i++)
      {
// Set the font to the PLAIN version
        g.setFont(new Font(fontList[i], Font.PLAIN, 12));
// Draw an example
        g.drawString("This is the " +
          fontList[i] + "font.", 5, startY);
// Move down a little on the screen
        startY += 15;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// Set the font to the BOLD version
        g.setFont(new Font(fontList[i], Font.BOLD, 12));
// Draw an example
        g.drawString("This is the bold" +
          fontList[i] + "font.", 5, startY);
// Move down a little on the screen
        startY + = 15;

// Set the font to the ITALIC version
        g.setFont(new Font(fontList[i], Font.ITALIC, 12));
// Draw an example
        g.drawString("This is the italic " +
          fontList[i] + "font.", 5, startY);

// Move down a little on the screen with some extra spacing
        startY + = 20;
}
}
}
```

22.5.2 FontMetrics 类

FontMetrics 类允许用户检查某个特定字体的各种字符度量。Graphics 类的 getFontMetrics 方法返回某个特定字体的 FontMetrics 实例：

```
public abstract FontMetrics getFontMetrics(Font f)
```

用户也可以获取当前字体的字体度量

```
public FontMetrics getFontMetrics( )
```

FontMetrics 的实例总是与某个特定的字体关联的。要想找出 FontMetrics 实例引用的字体，用户可以使用 getFont 方法：

```
public Font getFont( )
```

getAscent、getDescent、getLeading 和 getHeight 方法返回字体的各种高度

```
public int getAscent( )
```

该方法返回字体中字符的顶线高度。在这种字体中可能有些字符会超过这个顶线高度。

```
public int getDescent( )
```

该方法返回字体中字符的底线高度。在这种字体中可能有些字符会超过这个底线高度。

```
public int getLeading( )
```

该方法返回该字体的行距值。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public int getHeight( )
```

该方法返回字体的总高度,是顶线高度、底线高度和行距三者之和。

因为某些字符可能会超出正常的顶线和底线,所以用户可以使用 getMaxAscent 和 getMaxDescent 获取绝对限制:

```
public int getMaxAscent( )
public int getMaxDescent( )
```

字符的宽度通常是用术语"超前"(advance)给出。超前是指字符本身占用的空间加上字符后面的空白。打印在屏幕上的字符串的宽度是字符串中所有字符的超前的总和。charWidth 方法返回某个指定的字符的超前:

```
public int charWidth(char ch)
public int charWidth(int ch)
```

用户也可以使用 getMaxAdvance 方法获得字体中所有字符中最大的超前值:

```
public int getMaxAdvance( )
```

FontMetrics 类最常见的用途就是获得某串字符的宽度或超前。StringWidth 方法返回字符串的超前:

```
public int StringWidth(String str)
```

用户也可以获得字符数组或字节数组的宽度。

```
public int charWidth(char[ ] data , int offset, int len)
```

该方法返回存储在 data 中从 offset 开始的 len 个字节的宽度。

getWidths 方法返回由字体中前 256 个字符的宽度组成的数组。

```
public int [ ] getWidths( )
```

22.6 画模式

Graphics 类有两种不同的绘制图形的模式:绘画模式和异或模式。绘画模式意味着当图形被画出时,图形中的所有点覆盖原来该区域中的点。换句话说,如果用户画一条兰色的直线,那么这条线上的所有点都是兰色的。用户可能认为任何时候都是这样的,但是这也未必。其实还有另外一种绘制模式叫做异或模式(XOR),XOR 是 exclusive _ OR 的缩写。

异或绘图方式可以追溯几十年以前。为了解释异或模式的工作方式,请用户先暂时忘掉颜色处理,并想象着我们正在用白色在黑背景上绘画。当使用异或模式绘画时就涉到两个像素的结合问题,其中一个是用户要画的像素,另一个是屏幕上原来就有的像素。如果用户试图在当前为黑色的像素上画一个白像素,这时用户会得到一个白色像素点。如果用户试图在当前已经为白色的像素上画一个白色像素,那么用户将会得到一个黑色像素点。

这个概念听起来很奇怪,但这在使用异或手段的动画中则是司空见惯的事情。为了理解为什么,用户首先应该知道在异或模式下绘制一个图形之后,如果在异或模式再画一次这个图形,这将擦除掉该图形。如果用户在异或模式下移动一个图形,在用异或模式画出图形后,应在原位置用异或模式再画一次(擦除该图形),然后在新位置用异或模式重画。当两个对象覆

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

盖时，覆盖区看起来像是负片：黑色变成白色而白色变成黑色。用户可能不需要在动画中使用这种技术，但是用户至少可以从中得到一些启发。

> **注意**：当在彩色系统中使用异或模式时，可以把使用的颜色当作前面例子中的白色来看待，而将原有的颜色看做是异或色，或者是前面例子中的黑的背景色。因为这时颜色种类很多，所以异或方式的颜色结合也很有趣，但是用户仍然可以通过再画一次把已经画出的图形擦掉。

要想改变画模式为异或模式，只要调用 setXORMode 方法并传递给它想要使用的异或颜色即可。程序清单 22.9 中的小应用程序是一个简单的动画，它使用异或画法使圆球通过正方形。

程序清单 22.9 BallAnim.java 的源代码

```
import java.awt.*;
import java.applet.*;
import java.lang.*;

//
// The BallAnim applet uses XOR mode to draw a rectangle
// and a moving ball. It implements the Runnable interface
// because it is performing animation.

public class BallAnim extends Applet implements Runnable
{
    Thread animThread;
    int ballX = 0;      // X coordinate of ball
    int ballDirection = 0;  // 0 if going left-to-right, 1 otherwise

// Start is called when the applet first cranks up. It creates a thread for
// doing animation and starts up the thread.

    public void start()
    {
      if (animThread == null)
      {
        animThread = new Thread(this);
        animThread.start();
      }
    }

// Stop is called when the applet is terminated. It halts the animation
// thread and gets rid of it.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void stop()
    {
      animThread.stop();
      animThread = null;
    }

// The run method is the main loop of the applet. It moves the ball, then
// sleeps for 1/10th of a second and then moves the ball again.

    public void run()
    {
      Thread.currentThread().setPriority(Thread.NORM_PRIORITY);

      while (true)
      {
        moveBall();
        try {
          Thread.sleep(100); // sleep 0.1 seconds
        } catch (Exception sleepProblem) {
// This applet ignores any exceptions if it has a problem sleeping.
// Maybe it should take Sominex
        }
      }
    }

    private void moveBall()
    {
// If moving the ball left-to-right, add 1 to the x coord
      if (ballDirection == 0)
      {
        ballX++;

// Make the ball head back the other way once the x coord hits 100

        if (ballX > 100)
        {
          ballDirection = 1;
          ballX = 100;
        }
      }
      else
      {

// If moving the ball right-to-left, subtract 1 from the x coord
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            ballX * ;

    // Make the ball head back the other way once the x coord hits 0
            if (ballX < = 0)
            {
              ballDirection = 0;
              ballX = 0;
            }
          }

          repaint();
        }

        public void paint(Graphics g)
        {
          g.setXORMode(getBackground());
          g.fillRect(40, 10, 40, 40);
          g.fillOval(ballX, 0, 30, 30);
        }
    }
```

图 22.13 是 BallAnim 小应用程序运动中的一个快照。注意当球穿过正方形时颜色发生改变。



图 22.13 当对象发生接触时异或模式绘制产生相反的效果

22.7 绘制图像

Graphics 类提供了一种使用 drawImage 方法绘制图像的手段：

```
public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)
public abstract boolean drawImage(Image img,int x,int y,int width, int height, ImageObserver observer)
public abstract boolean drawImage(Image img, int x, int y, Color bg, ImageObserver ob )
public abstract boolean drawImage(Image img, int x, int y,int width, int height, Color bg, ImageObserver ob)
public abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, Color bg, ImageObserver ob)
public abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver ob)
```

drawImage 方法中的 observer 参数是一个负责监视图像实际画出的时间的对象。如果用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

户在自己的小应用程序中调用 drawImage 方法，用户可以把 this 作为观察器参数(observe r)传递，因为 Applet 类实现了 ImageObserver 接口。bg 参数用于指明背景区域的颜色，新的图像就绘制在该区域。如果图像拥有透明像素点，则透明像素点使用 bg 颜色绘制。

drawImage 方法可以绘制图像的一部分，同时还可以对图像进行缩放。sx 和 sy 参数指示源图像区域的左上角和右下角。dx 和 dy 参数指示要绘制的图像的左上角和右下角。如果源图像尺寸与要绘制的图像的尺寸不同，那么图像就需要作适当的缩放。

要想绘制图像，用户需要首先获得图像。但是 Graphics 类并不提供获得对象的方法。幸运的是，Applet 类提供的 getImage 方法可以用来获取图像。程序清单 22.10 获得并绘制了一幅图像。图 22.14 显示了这个小应用程序的输出。



图 22.14　用户可以在 Java 小应用程序中使用 drawImage 方法绘制任意的 GIF 图像和 JPEG 图像

```
import java.awt.*;
import java.applet.*;

//
// This applet uses getImage to retrieve an image
// and then draws it using drawImage

public class DrawImage extends Applet
{
    private Image samImage;

    public void init()
    {
      samImage = getImage(getDocumentBase(), "samantha.gif");
    }

    public void paint(Graphics g)
    {
      g.drawImage(samImage, 0, 0, this);
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

22.8 MediaTracker 类

用户可能面临的问题是当试图显示的图像是来自于速度较慢的网络连接(例如,14.4 千字节/秒的调制解调器)的图像。当用户开始绘制图像时,图像可能尚未全部获得。用户可以使用一个叫做 MediaTracker 的辅助类来确定图像是否已经准备好显示。

为了使用 MediaTracker,用户必须首先为小应用程序创建一个 MediaTracker。

```
public MediaTracker(Component comp)
```

该语句为指定的 AWT 组件创建了一个新的媒体跟踪器。comp 参数是典型地使用媒体跟踪器的小应用程序。

例如,下面的语句可以在小应用程序中创建媒体跟踪器:

```
MediaTracker myTracker = new MediaTracker(this);  //"this"引用小应用程序
```

接下来试图获得用户希望显示的图像

```
Image myImage = getImage("samantha.gif");
```

现在用户告诉 MediaTracker 用眼睛盯着图像。当用户增加一幅图像给 MediaTracker 时,用户也可以给出一个数字的 id:

```
public void addImage (Image image,int id )
```

id 值可以被用于多图像显示,当用户希望知道整个一组图像是否已经准备完毕,用户可以使用单个 ID 去检查它。如果用户在显示图像之前打算缩放它,用户应该在 addImage 调用中指明打算使用的宽度和高度:

```
public synchronized void addImage(Image image,int id ,int width,int height)
```

用户也可以简单地只给图像一个为零的 ID:

```
myTracker.addImage(myImage,0);  //跟踪图像,给出一个值为 0 的 id
```

一旦用户启动了图像跟踪,用户就可以加载图像并通过使用 waitForID 方法等待它准备完毕。

```
public void waitForID(int id)
```

该语句等待 ID 号为 id 的所有图像。

```
public void waitForID(int id ,long ms)
```

该语句等待 ID 号为 id 的所有图像,最多等待 ms 毫秒。

用户也可以使用 waitForAll 方法等待所有的图像:

```
public void waitForAll()
```

和使用 waitForID 方法一样,用户可以给出等待的最多毫秒数:

```
public void waitForAll(long ms)
```

用户可能不希望在启动小应用程序之前花时间去等待图像加载,可以使用 status 方法初

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

始化加载，但不等待加载。当用户调用 statusID 时，用户需要传递图像的 ID 和一个指示是否开始加载图像的布尔标志。如果用户传递了一个 true 值，那么该方法将开始加载图像：

```
public int statusID(int id ,boolean startLoading)
```

和 statusID 成对的另一个方法是 statusAll，statusAll 检查 MediaTracker 中所有图像的状态：

```
public int statusAll(boolean startLoading)
```

statusID 和 statusAll 方法都返回一个整数，该整数可以是下列标志中的任意一个：

- MediaTracker.ABORTED　如果一组图像中某一个加载失败
- MediaTracker.COMPLETE　如果一组图像中某一个加载完毕
- MediaTracker.LOADING　如果某些图像正在加载
- MediaTracker.ERRORED　如果某些图像在加载过程中遇到错误

用户也可以使用 checkID 和 checkAll 方法查看某个图像是否已经被成功地加载。所有 checkAll 方法的变体和 checkID 方法的变体都返回一个布尔值，如果被检查的图像已经被加载，则该布尔值为 true。

```
public boolean checkID(int id)
```

如果 ID 值为 id 的所有图像已经被加载完毕，则该方法返回 true。如果图像没有加载，那么不会因此而开始加载图像。

```
public synchronized boolean checkID(int id, boolean startLoading)
```

如果 ID 值为 id 的所有图像已经被加载完毕，该方法返回 true 值。如果 startLoading 为 true，那么该方法将开始加载尚未被加载的图像。

```
public boolean checkAll( )
```

如果被该 MediaTracker 跟踪的所有图像全部被加载完毕，则该方法返回 true，但是如果某个图像尚未被加载，该方法并不开始加载该图像。

```
public boolean checkAll(boolean startLoading)
```

如果被该 MediaTracker 跟踪的所有图像全部被加载完毕，则该方法返回 true。如果 startLoading 的值为 true，那么该方法将开始加载尚未被加载的图像。

程序清单 22.11 中的小应用程序使用 MediaTracker 监视某个图像是否加载完毕。该小应用程序在图像被加载之前在图像区域绘制文本；当图像加载完毕时，它将绘制图像。

[redacted]

```
import java.awt.*;
import java.applet.*;
import java.lang.*;
//
// The ImageTracker applet uses the media tracker to see if an
// image is ready to be displayed. In order to simulate a
// situation where the image takes a long time to display, this
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// applet waits 10 seconds before starting to load the image.
// While the image is not ready, it displays the message:
// "Image goes here" where the image will be displayed.

public class ImageTracker extends Applet implements Runnable
{
    Thread animThread;      // Thread for doing animation
    int waitCount;          // Count number of seconds you have waited
MediaTracker myTracker;     // Tracks the loading of an image
    Image myImage;          // The image you are loading

    public void init()
    {
// Get the image you want to show
    myImage = getImage(getDocumentBase(), "samantha.gif");

// Create a media tracker to track the image
    myTracker = new MediaTracker(this);

// Tell the media tracker to track this image
    myTracker.addImage(myImage, 0);
  }

    public void run()
    {
      Thread.currentThread().setPriority(Thread.NORM_PRIORITY);

      while (true)
      {
// Count how many times you've been through this loop
        waitCount++;

// If you've been through 10 times, call checkID and tell it to start
// loading the image
        if (waitCount == 10)
        {
          myTracker.checkID(0, true);
        }

        repaint();
        try {
// Sleep 1 second (1000 milliseconds)
          Thread.sleep(1000); // sleep 1 second
        } catch (Exception sleepProblem) {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
      }
    }

    public void paint(Graphics g)
    {
      if (myTracker.checkID(0))
      {
    // If the image is ready to display, display it
        g.drawImage(myImage, 0, 0, this);
      }
      else
      {
    // Otherwise, draw a message where you will put the image
        g.drawString("Image goes here", 0, 30);
      }
    }

    public void start()
    {
      animThread = new Thread(this);
      animThread.start();
    }

    public void stop()
    {
      animThread.stop();
      animThread = null;
    }
}
```

22.9 实用图形类

AWT包含了几个实用类,这些类只描述几何形状但不执行实际的绘制。前面介绍的Polygon类就是实用图形类中的一个。其它的实用图形类是Point类、Dimension类和Rectangle类。

22.9.1 Point类

Point描述了Java坐标空间中点的x,y坐标。AWT中有几个方法可以返回Point类的实例。用户也可以通过向Point的构造器传递x坐标和y坐标来创建Point类的实例:

```
public Point(int x, int y)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用户也可以创建未初始化的点,或者使用另一个 Point 对象初始化一个点:

```
public Point( )
public Point(Point p)
```

Point 对象的 x 坐标和 y 坐标是公共的实例变量:

```
public int x
public int y
```

这意味着用户可以直接操作 Point 对象的 x 值和 y 值。用户也可以使用 move 方法和 translate 方法修改 x 和 y 的值:

```
public void move (int newX,int newY)
```

该方法将点的 x 坐标和 y 坐标修改成 newX 和 newY。

```
public void translate (int xChange ,int yChange)
```

该方法把当前的 x 坐标增加 xChange,把当前的 y 坐标增加 yChange。

22.9.2 Dimension 类

Dimension 类描述宽度和高度,但不是在固定的点上。换句话说,两个矩形可以有相同的尺寸,但不一定在相同的坐标上。空的构造器创建宽度和高度均为 0 的尺寸:

```
public Dimension( )
```

用户也可以在构造中指定宽度和高度:

```
public Dimension(int width,int height)
```

如果用户希望复制一个已经存在的 Dimension 对象,用户可以将该对象传递给 Dimension 构造器:

```
public Dimension(Dimension oldDimension)
```

尺寸的宽度和高度是公共的实例变量,所以用户可以直接操作它们:

```
public int width
public int height
```

22.9.3 Rectangle 类

Rectangle 类描述的是 Point 类和 Dimensin 类的结合。Point 描述了矩形的左上角,Dimension 描述的是矩形的宽度和高度。用户可以通过向构造器传递 Point 和 Dimension 来创建 Rectangle 类的实例:

```
public Rectangle(Point p, Dimension d)
```

除了创建 Point 和 Dimension 以外,用户也可以向构造器传递点的 x 坐标和 y 坐标以及尺寸的宽和高:

```
public Rectangle(int x, int y, int width, int height)
```

如果用户希望 x 和 y 的值为 0,用户可以只使用宽度和高度创建矩形:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public Rectangle(int width,int height)
```

如果用户只向构造器传递一个点，那么此时矩形的高度和宽度就被设置为 0：

```
public Rectangle(Point p)
```

类似地，如果用户只传递尺寸，那么 x 和 y 就被设置为 0：

```
public Rectangle(Dimension d)
```

用户可以使用另一个 Rectangle 对象作为新的 Rectangle 对象的坐标和尺寸。

```
public Rectangle(Rectangle r)
```

用户可以使用空构造器，那么 x、y、宽度和高度都被设置为 0：

```
public Rectangle( )
```

x、y、宽度、高度变量都是公共的实例变量，所以用户可以直接操作它们：

```
public int x
public int y
public int width
public int height
```

像 Point 类一样，Rectangle 类也包含有 move 方法和 translate 方法，这两个方法用于修改矩形的左上角：

```
public void move (int new X,int new Y)
public void translate(int xChange , yChange)
```

与 move 和 translate 修改矩形左上角的点坐标的方式一样，setSize 和 grow 方法修改矩形的尺寸。

```
public void setSize (int newWidth,int newHeight)
public void grow(int widthChange,int heightChange)
```

setBound 方法用一个方法修改所有的 x、y、宽度和高度：

```
public void setBounds(int newX,int newY,int newWidth,int newHeight)
```

如果矩形包含指定的 x，y 点，那么 contains 方法返回 true：

```
public boolean contains(int x,int y)
```

intersection 方法返回某个矩形与另一个矩形共同包含的矩形区域：

```
public Rectangle intersection (Rectangle anotherRect)
```

用户可以使用 intersects 方法确定两个矩形是否相交：

```
public boolean intersects (Rectangle anotherRect)
```

union 方法类似于 intersection 方法，只不过 intersection 返回的是两个矩形的公共区域，而 union 返回被该矩形包含的最小矩形：

```
public Rectangle union(Rectangle anotherRect)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

add 方法返回包含当前矩形和另一个点的最小矩形

```
public Rectangle add(Point p)
public Rectangle add(int x、int y)
```

如果该点包含在当前矩形中，那么 add 方法将返回当前矩形。add 方法也可使用矩形做为参数，在这种情况下，add 方法与 union 方法的效果一样。

```
public void add(Rectangle anotherRect)
```

22.10 Color 类

用户可以回想一下自己小时候学过的有关基色的知识。实际上有两种基色。当用户使用蜡笔绘画时，用户实际上在处理颜料。基本的颜色是红、黄和蓝。用户可能知道一些典型的混合效果，例如红 + 黄 = 橙，黄 + 蓝 = 绿，蓝 + 红 = 紫。黑是由所有颜色混合在一起产生的颜色；而白色是没有使用颜料时的颜色。

对光的基色处理略有不同。光的基色是红、绿和蓝。一些基本组合是红 + 绿 = 棕(或者是黄，取决于亮度)，绿 + 蓝 = 深蓝(亮蓝)，红 + 蓝 = 品红(紫色)；光线的黑色概念与颜料的黑白正好相反，黑色是由没有光产生的，而白色是由三基色混合而成的。换句话说，红 + 蓝 + 绿(等量的) = 白。Java 使用的颜色模式叫做 RGB 颜色模式。

用户可以通过指明红光、绿光和蓝光的量来定义 RGB 模式下的颜色。用户既可以使用零到 255 之间的整数表示，也可以使用 0.0 到 1.0 之间的浮点数表示。表 22.1 显示了某些常用颜色的红绿蓝分量。

表 22.1 常用颜色和它们的 RGB 值

颜 色 名	红 色 值	绿 色 值	蓝 色 值
white(白)	255	255	255
Light Gray(浅灰)	192	192	192
Gray(灰)	128	128	128
Dark Gray(深灰)	64	64	64
Black(黑)	0	0	0
Red(红)	255	0	0
Pink(粉红)	255	175	175
Orange(橙)	255	200	0
Yellow(黄)	255	255	0
Green(绿)	0	255	0
Magenta(品红)	255	0	255
Cyan(深蓝)	0	255	255
Blue(蓝)	0	0	255

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用户可以用下列三种方式创建自定义颜色：

```
Color(int red,int green,int blue)
```

使用值介于 0 到 255 之间的红、绿和蓝色分量创建颜色。

```
Color(int rgb)
```

当使用红、绿和蓝介于 0 到 255 之间的值创建某个颜色时，这三个数被组合成了一个整数。位 16～23 存放的是红色值，位 8～15 存放的是绿色值，位 0～7 存放的是蓝色值。这些值通常是使用十六进制表示的，所以用户可以很容易看出颜色值。例如 0x12345 给出红色值 0x12(十进制的 18)，绿色值 0x34(十进制的 52)和蓝色值 0x56(十进制的 96)。注意每种颜色实际上只占用两个十六进制位。

```
Color(float red,float green,float blue)
```

该方法使用 0.0 到 1.0 之间的红、绿和蓝值创建颜色。

一旦用户创建了颜色，用户就可以在 Graphics 类中使用 setColor 方法改变绘制的颜色：

```
public abstract void setColor(Color c)
```

例如，假设用户希望使用粉红色绘画。粉红色的色值可以选为：红 255，绿 192，蓝 192。下面的 paint 方法将颜色设置为粉红色并用粉红色画出一个圆：

```
public void paint(Graphics g)
{
    Color pinkColor = new Color(255,192,192);
    g.setColor(pinkColor);
    g.drawOval(5,5,50,50);
}
```

用户不必总是人为地去创建颜色。Color 类提供了许多预定义颜色：

- Color.white
- Color.lightGray
- Color.gray
- Color.darkGray
- Color.black
- Color.red
- Color.pink
- Color.orange
- Color.yellow
- Color.green
- Color.magenta
- Color.cyan
- Color.blue

用户可以使用 getRed 方法、getGreen 方法和 getBlue 方法找出给定颜色的红绿蓝值：

```
public int getRed( )
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public int getGreen( )
public int getBlue( )
```

下面的代码片段首先设置了一种颜色,然后又取出了该颜色的红绿蓝值:

```
int redAmount, greenAmount, blueAmount;
Color someColor = new Color(0x345678)//红为 0x34,绿为 0x56,蓝为 0x78
redAmount = someColor.getRed( );//redAmount 现在等于 0x34
greenAmount = someColor.getGreen( );//greenAmount 现在等于 0x56
blueAmount = someColor.getBlue( );//blueAmount 现在等于 0x78
```

用户可以使用 darker 方法和 brighter 方法使颜色加深或变亮:

```
public Color darker( )
public Color brighter( )
```

这些方法都返回一个新的 Color 实例,其中的颜色比源色更深或更亮。源色保持不变。

22.11 裁剪

裁剪是一种图形系统中的技术,它防止在某个区域之外绘画。基本地,用户可以画出一个矩形区域,用户试图在该区域之外绘制的东西全部被"裁剪掉"。通常情况下用户的小应用程序是沿边缘裁剪的。换句话说,用户不能在小应用程序窗口之外绘画。用户不能增大裁剪区;即用户不能在小应用程序窗口之外绘画,但是用户可以进一步把可绘画区域限制在小应用程序窗口中的某个部分。要想设置用户的裁剪区边界,用户可以使用 Graphics 类中的 ClipRect 方法:

```
public abstract void clipRect(int x, int y, int width, int height)
```

用户可以使用 getClipBounds 方法查询 Graphics 对象的当前裁剪区:

```
public abstract Rectangle getClipBounds( )
```

程序清单 22.12 中的小应用程序将绘画区域缩减在左上角坐标为(10,10),宽为 60 个像素,高为 40 个像素的矩形之内,并在该裁剪区内绘制了一个实心圆。图 22.15 显示了这个小应用程序的输出。



图 22.15 ClipRect 方法缩减了绘画区并将超出绘画区的内容裁剪掉

```
import java.applet.*;
import java.awt.*;

//
// This applet demonstrates the clipRect method by setting
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
// up a clipping area and trying to draw a circle that partially
// extends outside the clipping area.
// I want you to go out there and win just one for the Clipper...

public class Clipper extends Applet
{
    public void paint(Graphics g)
    {
// Set up a clipping region
        g.clipRect(10, 10, 60, 40);

// Draw a circle
        g.fillOval(5, 5, 50, 50);
    }
}
```

ClipRect 方法只减小当前裁剪区。Java 1.1 以前的版本一旦缩减了裁剪区就没有扩展它的方法。在 Java 1.1 中增加了 setClip 方法，该方法既可以减小裁减区，又可以扩展裁剪区：

```
public abstract void setClip(int x, int y,int width, int height)
```

为了适应非矩形裁剪区域，Sun 公司增加了 Shape 接口和将 Shape 对象用作裁剪区的方法。目前的 Shape 接口只有一个方法：

```
public abstract Rectangle getBounds( )
```

用户可以使用 setClip 的这种变体设置带有实现了 Shape 接口的裁剪区：

```
public abstract void setClip(Shape regian)
```

因为裁剪区可以是非矩形的，所以 getClipeBounds 方法将无法胜任获取裁剪区的工作了。这时需要使用 getClip 方法以 Shape 对象的形式返回当前的裁剪区：

```
public abstract Shape getClip( )
```

虽然 Shape 接口可以允许用户创建非矩形裁剪区域，但是用户还不可以这样做。在 Shape 接口中定义的唯一的方法是返回矩形区域。Shape 接口还需要进一步扩展方能支持非矩形区域。

22.12　动画技术

用户可能已经注意到了当运行 ShapeManipulator 小应用程序时屏幕会发生闪烁。这是有意在编写程序时未消除这种闪烁，目的是想让用户知道闪烁的缺点。是什么引起的闪烁呢？一个主要的原因是形状是在用户面前重画的。因为用户看到了不断的重画，所以就产生了闪烁的效果。解决这个问题的技术叫做双缓冲区技术。

双缓冲区的思想是先创建一个脱屏图像，将要绘制的内容全部画在脱屏图像上。一旦画

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

完之后,用户就可以将脱屏图像拷贝到真正的屏幕上,因为这个速度很快,所以用户看到的画面是立即被更新的。

闪烁的其它原因来自于 update 方法,小应用程序缺省的 update 方法先清除绘图区,然后再调用用户的 paint 方法。用户可以通过使用直接调用 paint 方法的 update 覆盖缺省的 update 方法来消除由清屏引起的闪烁:

```
public void update(Graphacs g)
{
    paint(g)
}
```

警告:使用这种手段修改 update 方法有一定的危险。用户必须让小应用程序知道屏幕尚未被清除。如果用户使用双缓冲区技术,这就不会再成为问题,因为用户将使用脱屏图像完全代换绘图区。

用户可以很容易地对 ShapeManipulator 小应用程序作一些修改,使得它支持双缓冲区并取消清屏。在类声明的顶部,用户需要增加一个供脱屏绘制用的 Image 对象:

```
private Image offScreenImage;
```

接下来用户需要增加一行 init 方法初始化脱屏图像:

```
offScreenImage = createImage(size( ).width,size()height);
```

最后,用户可以创建一个不清除实际的绘画区的 update 方法,但是应该确保 paint 方法绘制脱屏图像并将脱屏区域的内容拷贝到屏幕上(如程序清单 22.13 所示)。

```
public void update(Graphics g)
{
// This update method helps reduce flicker by supporting off - screen drawing
// and by not clearing the drawing area first. It enables you to leave
// the original paint method alone.

// Get the graphics context for the off - screen image
    Graphics offScreenGraphics = offScreenImage.getGraphics();

// Now, go ahead and clear the off - screen image. It is O.K. to clear the
// off - screen image, because it is not being displayed on the screen.
// This way, your paint method can still expect a clear area, but the
// screen won't flicker because of it.

    offScreenGraphics.setColor(getBackground());

// You've set the drawing color to the applet's background color, now
// fill the entire area with that color (i.e. clear it)
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
        offScreenGraphics.fillRect(0, 0, size().width,
            size().height);

    // Now, because the paint method probably doesn't set its drawing color,
    // set the drawing color back to what was in the original graphics context.
        offScreenGraphics.setColor(g.getColor());

    // Call the original paint method
        paint(offScreenGraphics);

    // Now, copy the off-screen image to the screen
        g.drawImage(offScreenImage, 0, 0, this);
    }
```

22.13　打印

向打印机发送信息的能力是 Java 1.0 中最明显的疏漏之一。幸运的是，Java 1.1 使用 printJob 类提出了这个问题。

为了打印东西用户需要做的第一件事情是创建一个 PrintJob 对象的实例。用户可以使用 java.awt.Toolkit 中的 getPrintJob 方法做这件事情：

```
public abstract PrintJob getPrintJob(Frame parent, String jobname, Properties props)
```

正如用户可以看到的那样，打印工作是与 Frame 对象关联的。如果用户从小应用程序中执行打印任务，那么用户必须在调用 getPrintJob 之前首先创建 Frame 对象。一旦用户有了 PrintJob 对象，用户就可以通过调用 PrintJob 对象中 getGraphics 方法打印单独的页；getGraphics 方法创建了一个 Graphics 对象，用户可以在该对象上绘画，如下所示：

```
public abstract Graphics getGraphics( )
```

每个 Graphics 的新的实例都代表着一个单独的打印页。一旦用户打印完所有要打印的页，就可以调用 PrintJob 中的 end 方法结束打印工作：

```
public abstract void end( )
```

由 getGraphics 返回的 Graphics 对象与传递给用户的 paint 方法的 Graphics 对象是一致的。用户可以调用所用适用于 paint 方法的所有绘画方法。事实上用户可以打印当前屏幕上的图像，可以通过调用用户自己的 paint 方法来做到这一点。调用 paint 方法时应当使用由 getGraphics 方法返回的 Graphics 对象作为参数。一旦用户完成了在 Graphics 对象上的绘制工作，用户就可以通过调用 dispose 方法来完成该页的打印工作。

在打印信息时，用户经常需要知道页的分辨率，即每英寸上打印多少个像素点。PrintJob 对象里的 getResolution 方法就返回这类信息：

```
public abstract int getPageResolution( )
```

getPageDimension 方法返回以像素为单位的页宽度和页高度：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public abstract Dimension getPageDimension( )
```

某些系统和某些计算机首先打印最后一页。用户可以调用 lastPageFirst 方法获知是否是先从最后一页开始打印：

```
public abstract boolean lastPageFirst( )
```

程序清单 22.14 显示了与著名的"Hello World"程序对等的打印程序。

```
import java.awt.*;
import java.applet.*;

public class PrintHelloWorld extends Applet
{
    public void init()
    {

// First create a frame to be associated with the print job
        Frame myFrame = new Frame();
// Start a new print job
        PrintJob job = Toolkit.getPrintJob(myFrame, "Hello", NULL);

// Get a graphics object for drawing
        Graphics g = job.getGraphics();

// Print the famous message to the graphics object
        g.drawString("Hello World!", 50, 100);

// Complete the printing of this page by disposing of the graphics object
        g.dispose();

// Complete the print job
        job.end();
    }
}
```

按照现在的标准来看，由 Graphics 对象提供的绘画功能还很基本。这些功能最终将被 Java 2D API 所取代。Java 2D API 提供了更健壮的绘画模型。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 23 章　JFC——Java 基础类

本章内容：

23.1　Java 基础类
23.2　JFC:概览
23.3　Helloword
23.4　改进 Helloworld
23.5　使用 JFC 增加按钮
23.6　增加工具提示和图标
23.7　使用弹出式菜单
23.8　边
23.9　复选框和单选按钮
23.10　使用 checkBoxPanel 改变文本对齐方式
23.11　标签窗格
23.12　滑杠
23.13　进度条

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

23.1 Java 基础类

用户第一个要问的问题可能是:为什么需要使用基础类?使用基础类有什么好处?

要理解这个问题,需要简要地看一下 Java 的历史。在 Java 被刚刚创建的时候,用户界面是使用 AWT 包中的类开发的。用户曾经在第 19 章"Java.awt:部件"和第 20 章"深入研究异常和事件"中较系统地学习了 AWT 包中的一些类,当时可能就已经发现,在设计 AWT 时有许多限制。

为了帮助用户理解使用基础类的原因,让我们看一下 Java 最初的开发者们面临的决策。当时他们需要找到一种创建屏幕组件的手段,因为各种不同的计算机创建的组件,比如按钮都有些差别,所以找出通用的方法不是一件易事。在两台计算机上一个对一个地增加按钮是很容易的事情,但是若想在两台计算机上把按钮摆放在同一个位置并具有相同的尺寸和形状则需要做许多工作。

为什么会是这个样子呢?图形真是一个有趣的东西。Windows 的用户一定熟悉 800 × 600 和 640 × 480这两种屏幕设置,Macintosh 的用户可能比较熟悉 1180 × 900 的屏幕设置。Solaris 的用户可能熟悉 1600 × 1280 的屏幕设置。现在让我们撕开这层表面考虑一下如何使得在 Macintosh 上创建的按钮在 Windows 上具有相同的外观效果。应该记住在各种不同的平台上按钮是有差异的。现在让我们看一下 Macintosh 中的 Netscape 4.0 与 PC 上的 Netscape 4.0 的差别。图 23.1 是 Macintosh 上的 Netscape 4.0,图 23.2 是 PC 机上的 Netscape 4.0。

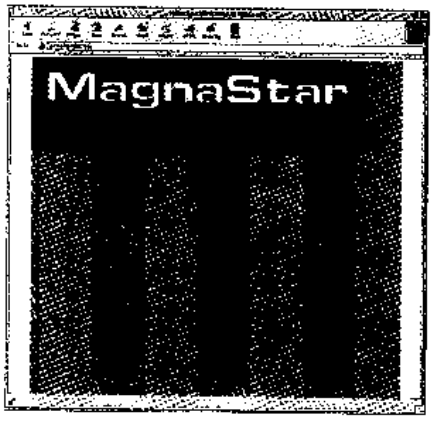


图 23.1 Macintosh 上的 Netscape 4.0

如果用户看一下两个机器上的相应按钮就会发现它们的差别。为什么会是这样?这是一种概念,叫做外观和感觉。人们为 Macintosh 编写的软件中的按钮在 Macintosh 机器上看是一个样子,当 Windows 用户使用这些按钮时看起来又是另一个样子。

为了解决这个问题,Java 的设计者们决定使用设计图案,这样一来 Java 程序访问的是抽象的按钮,不是具体系统里的看得见感觉得到的按钮。这意味着当用户将 java.awt.Button 组件摆放到屏幕上时,用户实际上使用的是具有本地上的外观和感觉的那种按钮。这样就可以使得 Windows 机器上创建的按钮在 Windows 上和在 Macintosh 上看起来和用起来感觉一样了。同样在 Macintosh 机器上创建的按钮在 Windows 上和在 Macintosh 上看起来和用起来的感觉也是一样的。看起来像是找到了感觉了,对吗?

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

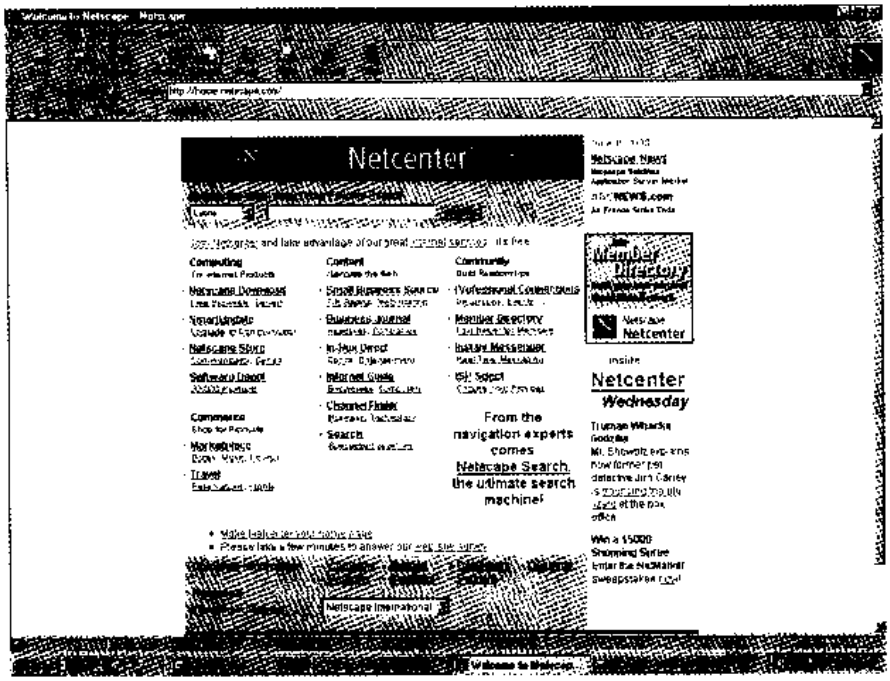


图 23.2　PC 上的 Netscape 4.0

但遗憾地是，事情远非如此简单，并且在很久以前 Java 的实现就不再是从一种计算机系统向另一种系统上的移植。

用户可以看到创建一个真正的可以在各个系统中都能一致工作的抽象系统是非常困难的事情。例如，并非所有的系统都按照相同的方式同时报告鼠标的移动。并非所有系统都显示相同的字符，并且各种平台都有自己特色的东西，比如文本区的变化。正是因为这些原因和这些原因的变体，使得编写可以在各种平台上通用(使用方法和外观一致)的 AWT 系统成为难乎其难的事情。

因为不是所有的 AWT 系统都是一致的，所以像你我这样的人在创建应用程序时必须在不同的计算机上运行大量的试验例子，以期望真正兑现 Sun 公司的“写一次就可以到处用”的承诺。

不幸的是，这种大量的试验完全可以把 Java 所提供的开发时间短的好处统统吃掉。所以几个开发者决定建立一个不同的系统种类。新的系统只依赖于最通用的组件——容器(面板之父)。因为容器在各种系统里是完全一致的，用户可以不依赖任何平台来绘画。Microsoft 公司的解决方案是应用基础类(Application Foundation Classes，简称 AFC)，Netscape 公司的解决方案是国际互联网基础类(Internet Foundation Class，简称 IFC)，其它公司还有很多独立的解决方案。每个解决方案都有一个非常大的限制因素：文件大小。要想创建完整的新的窗口化系统必然要占用很大的磁盘空间。如果在硬盘上，600K 的空间肯定算不上大。但是如果用户必须经过国际互联网下载，那么 600K 就显得大了。除此之外，没有一个支持某个基础类的标准体，许多开发者都面临着选择解决方案的问题，因此，在 1997 年春天，Sun 公司与 Netscape 公司合作推出了 JFC。遗憾的是 Microsoft 为了他自己的 WFC 解决方案而决定不采用 JFC，但是有希望在不远的将来会看到他们采用 JFC。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

23.2 JFC:概览

JFC系统是基于两个原始的东西的:第一个是AWT的容器和框架组件,第二个是JDK 1.1事件模式。因为JDK 1.1事件模式的缘故,IFC用户切换到JFC比AWT用户更困难。

23.2.1 设置JFC

JFC是JDK 1.2内核的一部分,但是它可以加进JDK 1.1中。用户可以从Sun公司的站点上下载JFC包,地址是:

http://www.javasoft.com/products/jfc/

当用户下载完JFC API文件后,用户需要对文件执行解压缩操作。压缩文件中包含有几个.jar文件。除了包含核心swing类的swing.jar文件以外,用户还能够找到几个面向外观和感觉的.jar文件,比如Windows.jar和jlf.jar文件。

为了方便使用所有这些.jar文件,Sun公司增加了一个环境变量。swing-home变量用于指向安装了swing包的目录。例如,在Windows机器上,用户可能需要输入下列内容:

```
set swing_home=c:\swing
```

23.3 HelloWorld

和我们在本书中的一惯做法一样,我们现在也将通过创建最简单的应用程序开始介绍JFC。在程序清单23.1中,用户可以看到HelloWorldJFC.java的源代码。当用户编译并运行HelloWorldJFC时,用户应该看到如图23.3所示的结果。

```
import java.awt.*;
import java.awt.eent.*;
import com.sun.java.swing.*;

public class HelloWorldJFC extends JComponent {
    static JFrame myFrame;

    public void paint(Graphics g){
        g.setColor(Color.black);
        g.drawString("HelloWorld",20,15);
    }
    public static void main(String args[]){
        myFrame = new JFrame("Hello World!");
        HelloWorldJFC jt = new HelloWorldJFC();
        myFrame.getContentPane()/and("Center",jt);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        myFrame.setSize(100,50);
        myFrame.setVisible(true);
    }
}
```



图 23.3　HelloWorldJFC 应用程序

23.3.1　编译 HelloWorldJFC

用户可以像编译其它的 Java 程序那样编译 HelloWorldJFC，只要 swing.jar 文件被包含在用户在类路径里就可以。对于某些使用 IDE 的用户，可能需要把该文件从 jar 档案中取出来以便在编译时使用它。

23.3.2　运行中的 HelloWorldJFC

正如用户可能猜到的那样，运行 HelloWorldJFC 与运行任何其它的 JAVA 程序一样，只要将 swing.jar 放在用户的类路径中即可。可以简单地输入下列命令：

```
java HelloWorldJFC
```

为了使该程序能够运行，用户必须执行一个额外的步骤：包含该平台的“外观和感觉 .jar”文件。在 windows 平台下，这意味着必须将 windows.jar 包含进去，在 UNIX 机器上 motif.jar 将起这种作用，在 Macintosh 机器上需要包含 mac.jar 文件。当然也可以使用 SwinAll.jar 代替上述 .jar 文件。这个文件包括所有标准的外观和感觉。如果用户没有包含“外观和感觉 .jar”文件，用户将获得如下错误：

```
java.lang.Error: can't load com.sun.java.swing.Windows.WindowsLookAndFeel at com.sun.java.swing.
        UIManager.initializeDefaultLookAndFeel(UIManager.java:318)
        at com.sun.java.swing.UIManager.maybeInitialize(UIManager. java:386)
        at com.sun.java.swing.UIManager.maybeInitialize(UIManager. java:395)
        at com.sun.java.swing.UIManager.getDefaults(UIManager. java:146)
        at com.sun.java.swing.UIManager.getColor(UIManager. java:155)
        at com.sun java.swing.JPanel. < init > (JPanel.java:50)
        at com.sun java.swing.JPanel. < init > (JPanel.java:83)
        at com.sun.java.swing.jRootPane.createGlassPane(JRootPane.java:145)
        at com.sun.java.swing.JRootPane. < init > (JRootPane.java:112)
        at com.sun.jave.swing.JFrame.creatRootPane(JFrame.java: 105)
        at com.sun.iava.swing.JFrame.frameInit(JFrame. java:99)
        at com.sun.java.swing.JFrame. < init > (JFrame.java:93)
        at HelloWorldJFC.main(HelloWorldJFC. java:41)
```

如果用户看到这个错误，那么可能是因为在类路径中没有增加 Windows 外观和感觉（或者有问题的那一个）。因为单独的外观和感觉被打包存放在单独的 JAR 文件中，所以，所有需要用户做的事情就是增加需要的 JAR 文件到类路径中。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

23.3.3 理解 HelloWorldJFC

正如用户看到的那样，HelloWorldJFC 实际只包含有两个方法：paint 方法和 main( )方法。如果用户看过 paint( )方法，可能会发现，它与在第 15 章“高级小应用程序代码”中的 paint 方法几乎是一样的。

这里的 main( )方法则包含了一些不同的功能。

这里创建的框架与在 AWT 应用程序中创建的框架不同，该框架不是由 java.awt.Frame 创建的，它是由 JFrame 创建的。JFrmae 与 Frame 只有一种主要的差异：它提供了几个分层的窗格，而不是一个平的面板。

23.3.4 窗格分层

JFC 使用了多个层摆放组件。这种分层允许用户重迭组件和在组件顶上绘画。因为分层模式是建立的 JFC 系统内部的，所以用户不像使用 AWT 那样会获得不一致的结果，例如，如果用户希望把工具提示放在按钮上并且希望工具提示直接出现在按钮下方，不管有没有东西在这个位置显示，用户可以简单地画在玻璃窗格上。在 AWT 中这是不可能的，因为当某个按钮占据了屏幕某个位置，它一般就不允许再在上面绘制别的东西了。

JFC 包含了许多层，图 23.4 显示了面板视图和它们的顺序。

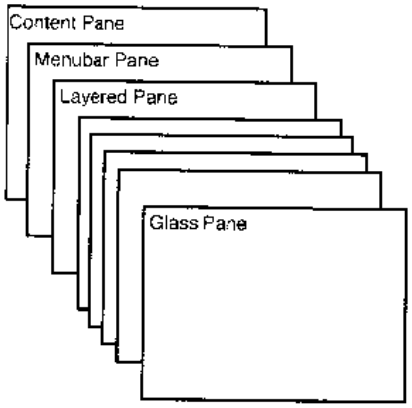


图 23.4 JFC 分层摆放了几个窗格

所有这些在 HelloWorld 示例的 main 方法中全都包含，它不是简单地增加了面板，而是将面板加在内容窗格中：

```
myFrame.getContentPane( ).add("Center", jt);
```

通常讲，在大部分用户向 JFC 容器增加组件时，将把它加进内容窗格中。如果需要覆盖其它组件，就需要把它加到分层的窗格上或玻璃窗格上。

23.4 改进 HelloWorld

在原来的 HelloWorld 版本中有两个需要改进的东西。首先，如果用户实际地运行该程序，会注意到当用户按下关闭窗口按钮(在 Windows 中就是在窗口右上角的 x 按钮)时程序并没有实际地退出。就像标准的 AWT 那样，用户需要创建一个窗口适配器来监听 windowClosing 事件并退出程序。第二件用户关心的事情是替换掉使用组件(标签)显示“Hello World”文

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

本，目前程序中可以使用 paint( )方法直接在屏幕上绘制字符串。虽然这是与小应用程序的 HelloWorld 类似的观点，但这不是最好的做法。在程序清单 23.2 中，用户可以在改进过的 HelloWorldJFC 应用程序中找到这个代码。当运行 HellowWorldJFC2 时它会显示图 23.5 所示的结果。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;

public class HelloWorldJFC2 extends JPanel {
  static JFrame myFrame;
  public HelloWorldJFC2(){
    JLabel label = new JLabel ("Hello World!");
    add(label);
  }

  public static void main(String args[]){
    myFrame = new JFrame("Hello World!");
    HelloWorldJFC2 hello = new HelloWorldJFC2();
    myFrame.getContentPane().add("Center",hello);
    myFrame.setSize(200,100);
    myFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {System.exit(0);}
      });
    myFrame.setVisible(true);
  }
}
```



图 23.5　HelloWorldJFC2.java 应用程序

像 AWT 包中的 Container 类一样，HelloWorldJFC2 使用了 add()方法增加组件。在本示例中，构造器中增加的组件是 JLabel。

23.4.1　JLabel

HelloWorldJFC2 的构造器中的 JLabel 用于显示“Hello World”文本。在程序清单 23.2 中，JLabel 并不提供比 AWT 中相应的标签更多的功能。但是 JLabel 与 Label 有两个差异。

首先 JLabel 像大部分 JFC 组件一样，既可以支持文本也可以支持图像（或两者同时出现）。所以，虽然我们在 HelloWorld 示例中没有使用它，但是我们可以增加一个图标使它与文

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

本同时显示。要做到这一点可以使用两种方式：

- 使用容纳该图标的构造器之一。
- 使用 setIcon() 方法添加图标。

23.4.2 增加图标

图标是 JFC 中的一种图形表示。图标可以是图像，也可以是由程序控制的图画。要想在任何 JFC 集中的组件上增加图标，用户可以使用下列两种技术之一：

- 指定构造器中的图标。
- 稍后使用 setIcon( )方法设置图标

在我们的 HelloWorldJFC2 示例中，如果我们有一个名为 feet.gif 的图像并且希望该图像和"Hello world"文本一起显示，那么我们就可以使用 ImageIcon 类把该图像加进去。

为了做到这些，让我们把程序清单 23.2 中的构造器修改成下面这个样子：

```
public HelloWorldJFC2( ){
    Icon icon = new ImageIcon("feet.gif");
    JLabel label = new JLabel("Hello World!", icon, SwingConstants.RIGHT);
}
```

图 23.6 显示了修改后的结果。



图 23.6 标签既可以有文本也可以有图标

注意：除了指定标签文本和图标之外，JLabel 还需要知道这两个项目的摆放关系。稍后在程序清单 23.7 中我们将继续深入地讨论如何使用对齐规则。

如果用户决定在创建图标之后指定图标，那么用户可以使用 setIcon 方法设置图标，在这种情况下用户可以把构造器调用修改成如下所示的样子：

```
public HelloWorldJFC2( ){
 Icon icon = new ImageIcon("feet.gif");
 JLabel label = new JLabel("Hello world");
 label.setIcon(icon);
  add(label);
}
```

23.4.3 关闭窗口

HelloWorldJFC2 中包含有额外的用于关闭窗口的代码，和 AWT 的 Fram 类一样，JFrame 类可以在窗口发生情况时产生事件，比如窗口被关闭，激活窗口，图标化窗口，打开窗口。如果 WindowListener 使用该框架在系统中注册过，那么这些事件将被送向 WindowListener。

在 HelloWorldJFC2 的 main( )方法中，用户可以看到下列代码，该代码增加了一个内置的类，它是当场创建的。在本例中，当窗口的关闭按钮被按下时，我们可以简单地退出程序。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
MyFrame.addwindwosListener(new windowAdapter(){
    public void windowClosing(WindowEvent e){
        System.exit(o);
    }
});
```

23.5　使用 JFC 增加按钮

和 JLabel 一样，JFC 的按钮组件也有了改进。JFC 的按钮(JButton)增加了快捷键功能，在编程上它模拟单击动作；JFC 按钮可以使用字符串和图标，并且还有比 AWT 按钮更多的状态。程序清单 23.3 显示了两个按钮的示例(见图 23.7)。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;

public class ButtonExample extends JPanel {
  static JFrame myFrame;
  JLabel label;

  public ButtonExample(){
    label = new JLabel ("Hello World!");
    JButton hello = new JButton("Hello");
    hello.setMnemonic('h');
    hello.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
      System.out.println("Hello World!");
      label.setText("Hello World!");
      }
      });
    JButton bye = new JButton("Bye*);
    bye.setMnemonic('b');
    bye.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
      System.out.println("Bye World!");
      label.setText("Good Bye World!");
      }
      });
    add(bye);
    add(hello);
    add(label);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }

    public static void main(String args[]){
        myFrame = new JFrame("Button Example");
        ButtonExample jt = new ButtonExample();
        myFrame.getContentPane().add("Center",jt);
        myFrame.setSize(300,70);
        myFrame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {System.exit(0);}
        });
        myFrame.setVisible(true);
    }
}
```



图 23.7 按钮示例,使用 Java 的 L&F 的按钮

23.5.1 理解 ButtonExample

ButtonExample 的 main 方法与程序清单 23.1 和 23.2 中的 main 方法几乎是一样的,没有什么出奇之处。

本例中有趣的代码是 ButtonExample 构造器中的代码。在本例中我们增加了两个按钮和一个标签,该标签跟随着按钮的按动发生变化。

因为在程序清单 23.2 的 HelloWorldJFC2 示例中,用户已经看到了 JLabel 是如何工作的,所以在本例中我们着重介绍这两个按钮中的一个 Hello 按钮。创建 JButton 的代码非常类似于创建 java.awt.Button 的代码。

```
JButton hello = new JButton("Hello");
```

23.5.2 设置快捷键和助记符

在 HelloWorldJFC2 中接下来的一行是为按钮建立助记符,助记符是与快捷键对等的键。这意味着该应用程序的使用者可以使用他的鼠标按 Hello 按钮,或者通过按组合键来按按钮。在本例中,组合键是 Alt-H。所有的按钮都可以使用 SetMnemonic( )方法设置它们的助记符:

```
hello.setMnemonic('h');
```

23.5.3 监听来自于按钮的动作

当按钮被按下时或它的助记符键被按下时,按钮就产生一个 ActionEvnt 事件。为了"听到"这个动作,需要为按钮增一个 ActionListener 并创建一个内置的类,该类当按钮被按下时执行用户希望产生的动作。

```
hello.addActionListener(new ActionListener( ){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        public void actionPerformed(ActionEvevt ae){
          System.out.println("Hello World!");
          label.setText("Hello World!")
        }
    });
```

23.6　增加工具提示和图标

使用分层视的一个优点是增加了 JFC 的用途，比如工具提示。工具提示可以被加进任何 JComponent 中，包括 JButtons 。

按钮的标签通常是按钮实际功能的缩写。这一点可以在程序清单 23.3 的 ButtonExample 中看到。

和 JLabel 一样，JButton 可以使用图标作为标签的一部分，所以程序清单 23.4 中修改过的 ButtonExample 在按钮上增加了工具提示，并且把 feet.gif 图像增加到 Hello 按钮上（如图 23.8 所示）。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;

public class TipButtons extends JPanel {
  static JFrame myFrame;
  protected JLabel label;

  public TipButtons(){
    label = new JLabel ("Hello World!");
    label.setOpaque(true);
    JButton hello = new JButton("Hello");
    hello.setMnemonic('b');
    hello.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
      label.setText("Hello World! *);
      }
      });
    //Set the ToolTip for the hello button
    hello.setToolTipText("Select to change label to Hello World");

    JButton bye = new JButton("Bye");
    bye.setMnemonic('b');
    bye.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            label.setText("Good Bye World!");
            }
            });
        //Set the ToolTip for the bye button
        bye.setToolTipText("Select to change label to Good Bye World");

        add(bye);
        add(hello);
        add(label);
    }

    public static void main(String args[]){
        myFrame = new JFrame("Tooltiped Buttons");
        TipButtons tb = new TipButtons();
        myFrame.getContentPane().add("Center",tb);
        myFrame.setSize(300,75);
        myFrame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {System.exit(0);}
            });
        myFrame.setVisible(true);
    }
}
```



图 23.8 当用户的鼠标经过有工具提示设置的按钮时，将有提示文本显示以帮助用户了解该按钮的功能

23.7 使用弹出式菜单

类似于工具提示的 JFC 设计的另一个优点是增强的弹出式菜单功能。它允许用户使用鼠标单击并弹出一个额外的菜单，如图 23.9 所示。程序清单 23.5 显示的就是这个示例。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;

public class PopupExample extends JPanel {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
static JFrame myFrame;
protected JLabel label;
JPopupMenu popup;

public PopupExample(){
  label = new JLabel ("Hello World!");
  label.setOpaque(true);
  add(label);

  popup = new JPopupMenu();

  //create the first menu item
  JMenuItem menuItem1 = new JMenuItem("Hello World!");
  menuItem1.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ae){
    label.setText("Hello World!");
    }
    });

  //create the second menu item
  JMenuItem menuItem2 = new JMenuItem("Good Bye World!");
  menuItem2.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ae){
    label.setText("Good Bye World!");
    }
    });
  //add the menu items to the popup menu
  popup.add(menuItem1);
  popup.add(menuItem2);

  addMouseListener(new MouseAdapter(){
    public void mouseReleased(MouseEvent evt){
              //Pop up the menu at the location where the mouse was pressed
              if (evt.isPopupTrigger()){
                popup.show(evt.getComponent(),evt.getX(),evt.getY());
      }
    }
  });
}

public static void main(String args[]){
  myFrame = new JFrame("Popup Example");
  PopupExample example = new PopupExample();
  myFrame.getContentPane().add("Center",example);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        myFrame.setSize(300,75);
        myFrame.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e) {System.exit(0);}
          });
        myFrame.setVisible(true);
    }
}
```



图 23.9 弹出式菜单——当用户单击鼠标右键时会出现一个弹出式菜单

23.7.1 理解 PopupExample

创建和显示一个 JFC 弹出式 菜单有三步骤。第一步是创建弹出式菜单本身。

```
popup = new JPopupMenu( );
```

第二步是创建在菜单中显示的菜单项。在 PopupExample 中,用户增加了两个菜单项。并且为每个菜单项都创建并增加一个 ActionListener。

```
JMenuItem menuItem1 = new JMenuItem("Hello World!");
menuItem1.addActionListener(new ActionListener( ){
    public void actionPerformed(ActionEvent ae){
        label.setText("Hello World!");
    }
});
```

注意:JMenuItem 是一个 JComponent(因为它是从 JComponent 扩展来的),所以像所有其它的 JComponent 一样,JMenuItem 上也可以增加工具提示。所以如果用户愿意,可以在每个菜单项上也加上工具提示。

最后一步是实际地弹出菜单。在本例中我们希望在弹出触发钮(在大部分平台上是鼠标右键)被按下时弹出该菜单。由 processMouseEvent( )方法负责处理弹出菜单的工作:

```
addMouseListener(new MouseAdapter( ){
public void mouseReleased(MouseEvent evt){
//在鼠标键按下处弹出该菜单
    if(evt.isPopupTrigger( ) ){
        popup.show(evt.getComponent( ),evt.getX( ),evt.getY( ) );
    }
  }
});
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

23.8　边

JFC 为所有组件增加的独有特性之一是增加可调整的边。任何对象都可以增加一个任意的边。所以如果用户想让按钮有一个花边或让标签有一个蚀刻边，那么用户就可以为它们加上这些边。

在 JFC 集中用户可以发现一个名为 com.java.swing.border 的包(Java 1.2 核心中的 java.swing.border)。这个包中包含许多不同的边，这些边都可以应用在各种抽象组件中。

清单 23.6 显示了来自清单 23.3 中具有不同边的两个按钮(见图 23.10)。

程序清单 23.6　BorderButtons.java——使用有边组件的示例

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.border.*;

public class BorderedButtons extends JPanel {
  static JFrame myFrame;
  protected JLabel label;
  JPopupMenu pm;

  public BorderedButtons(){
    label = new JLabel ("Hello World!");
    label.setBorder(new EtchedBorder());

    JButton hello = new JButton("Hello");
    hello.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
      label.setText("Hello World!");
      }
    });

    //set the border to an image. Note: the image will be tiled.
    Icon icon = new ImageIcon ("feet.gif");
    hello.setBorder(new MatteBorder(10, 10, 10, 10, icon));

    JButton bye = new JButton("Bye");
    bye.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
      label.setText("Good Bye World!");
      }
    });
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        bye.setBackground (SystemColor.control);
        bye.setBorder(new LineBorder(Color.green));

        add(bye);
        add(hello);
        add(label);
    }

    public static void main(String args[]){
        myFrame = new JFrame("Border Example");
        BorderedButtons jt = new BorderedButtons();
        myFrame.getContentPane().add("Center",jt);
        myFrame.setSize(300,75);
        myFrame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {System.exit(0);}
            });
        myFrame.setVisible(true);
    }
}
```



图 23.10 带边的按钮和标签

23.8.1 理解 BorerButtons

正如用户看到的那样，程序清单 23.6 与本章中另一个示例非常相似。关键的细节在于本例中的按钮和标签都有它们自己的边。它们每一个都有不同的边。在标签中使用蚀刻边。

```
label.setBorder(new EtchedBorder( ));
```

图为 JCommponent 拥有 setBorder( )方法，所以通过使用 setBorder( )方法可以使任何 JComponent 都可以拥有它自己的边集。

23.8.2 更多的边

除了可以设置特殊的边以外，某些边还可以层叠。正如用户在程序清单 23.6 的 BorderButtons 示例中看到的那样，hello 按钮使用了 MatteBorder。

```
hello.setBorder(new MatteBorder(10,10,10,10,icon));
```

MatteBorder 使用图标作为参数并将它显示在边内。对于许多人来说会觉得太靠近文本了。

用户可以通过使用 CompoundBorder 解决这个问题。CompoundBorder 可以将两个边层叠在一起，例如：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
hello.setBorder(new CompoundBounder(new MatteBorder(10,10,10,10,icon), new EmptyBorder(10,10,
10, 10)));
```

在这种情况下将边设置在按钮上的方法是 CompoundBorder。CompoundBorder 是由两条单独的边组成的,其中一个在另一个的里面。

EmptyBorder 基本上就一块空白。在程序清单 23.6 的按钮中使用这种边可以在文本的四周留出更多的空间,如图 23.11 所示。



图 23.11　带有更宽的边的加边按钮

还有一些其它的边也允许层叠。换句话说,用户可以组合使用这些边。另一个好的示例是 TitleBorder。TitleBorder 在自己的边内显示文本。如果用户想在 Hello 按钮的边中增加一个"hi"标题,那么用户可以把设置边的几行语句改成下面这样:

```
LineBorder lb = new LineBoredr(Color.green);
hello.setBorder (new TitleBorder(lb,"hi"));
```

在 borders 包中有许多边。每种边的外观都有一些细小的差别。图 23.12 显示了该包中的几个边。

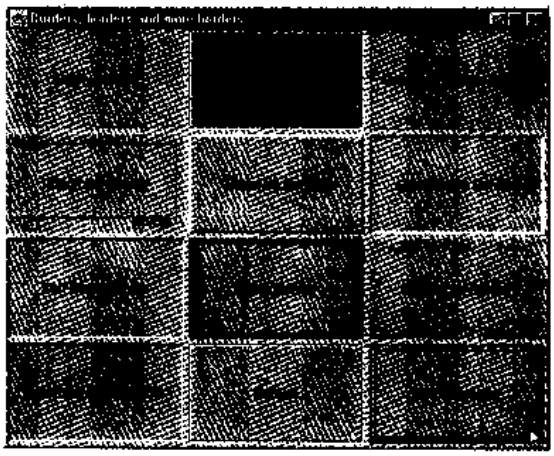


图 23.12　JFC 有许多标准的边,用户可以选择使用它们

23.9　复选框和单选按钮

在 JFC 里,复选框和单选按钮是非常相似的。唯一真正的不同是它们的外观不同。和 AWT 一样,JFC 复选框和单选按钮有两种不同的形式。单个的复选框相当于一个切换按钮。但是当多个复选框编组使用时,这些复选框就变成了选择"池",如图 23.13 所示。这个组类在 JFC 中叫做 ButtonGroup(与 AWT 中的 CheckBoxGroup 对应)。这个更加通用的组是以抽象按钮的方式工作的。抽象按钮意味着它可以用作 JFC API 中任何按钮。程序清单 23.7 显示了一组复选框的用法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 23.7 CheckBoxPanel.java——使用 JRC 复选框

```
public class CheckBoxPanel extends JPanel implements SwingConstants{
  public CheckBoxPanel(ActionListener al){

    Box vertBox = Box.createVerticalBox();

    Box topBox = Box.createHorizontalBox();
    Box middleBox = Box.createHorizontalBox();
    Box bottomBox = Box.createHorizontalBox();

    ButtonGroup group = new ButtonGroup();
    //Create the checkboxes
    JCheckBox north = new JCheckBox("North");
    north.addActionListener(al);
    north.setActionCommand("North");
    group.add(north);
    topBox.add(north);

    JCheckBox west = new JCheckBox("West");
    west.addActionListener(al);
    west.setActionCommand("West");
    group.add(west);
    middleBox.add(west);

    JCheckBox center = new JCheckBox("Center");
    center.addActionListener(al);
    center.setActionCommand("Center");
    group.add(center);
    middleBox.add(center);

    JCheckBox east = new JCheckBox("East");
    east.addActionListener(al);
    east.setActionCommand("East");
    group.add(east);
    middleBox.add(east);

    JCheckBox south = new JCheckBox("South");
    south.addActionListener(al);
    south.setActionCommand("South");
    group.add(south);
    bottomBox.add(south);

    vertBox.add (topBox);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        vertBox.add (middleBox);
        vertBox.add (bottomBox);

        add(vertBox);
    }
}
```

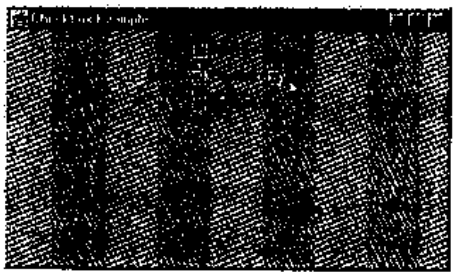


图 23.13　一组中的五个复选框

23.9.1　理解 CheckBoxPanel

程序清单中的 CheckBoxPanel.java 显示了如何建立一组复选框。注意每个复选框都被加入到 GroupBox 变量组中。

23.9.2　使用 ActionListeners 和设置事件

CheckBoxPanel 的构造器只使用一个参数 ActionListener。注意该动作监听器在程序清单 23.7 中用 addActionListener( )方法为每个复选框注册过。这个注册与用户在程序清单 23.3 中为 JButton 所做的注册是一致的。在本例中用户只是使用被传递给方法的 ActionListener 对象。

另外需要注意的是，在本例中我们显式地指明了动作命令，用的是 setActionCommand( ) 方法，而不是依赖缺省的动作。

23.9.3　使用箱和箱式布局

用户可能已经注意到我们在本例中使用了一个新类：Box。使用 JFC，用户可以获得一个新的布局管理器叫做箱式布局。箱式布局类似于网格布局类 GridLayout，但与网格不同的是箱式布局或者是水平对齐或者垂直对齐，但是不支持多列和多行。箱式布局不假设所有组件在主轴上拥有相同的尺寸。

在本例中使用的 Box 类是一个使用箱式布局的面板。它可以方便地创建垂直和水平的“长条”，在这个“长条”上我们可以放置组件。

在本例中创建了三个水平长条，每一个长条是一个“行”，并且这些长条一起被加进了垂直长条（如图 23.14 所示）。

用户会注意到每个复选框都被加进了一个箱中，例如：

```
topBox.add(north);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

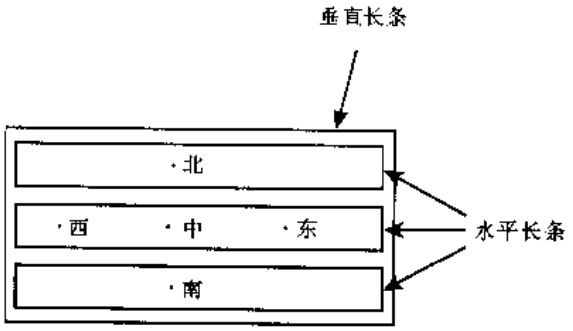


图 23.14 如何使用每个箱

23.10 使用 CheckBoxPanel 改变文本对齐方式

当用户在阅读本章前面的 JLabel 时，已经了解到如何同时使用图标和文本标签，用户可以指示图标向文本对齐。现在让我们看一下如何改变这种对齐方式。

JLabel 和 JButton 都包含两个指定文本位置和图标位置的方法：

```
setVerticalTextPosition( )
setHorizontalTextPosition( )
```

这两个方法都使用来自 SwingConstants 接口的参数。SwingConstants 包含许多常量。比如：TOP、BOTTOM、LEFT 和 CENTER。正如用户猜到的那样，当将文本位置设置为 TOP 时文本出现在图标上方。

在程序清单 23.7 中用户看到过如何把一组复选框放在一个组上。当每个按钮被按下时，它都产生一个 ActionEvent，该事件给出了方向。现在我们使用 CheckBoxPanel 来改变按钮上的文本和图标的对齐方式，如程序清单 23.8 和图 23.15 所示。

程序清单 23.8 CheckBoxPanel 改变对齐方式

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.border.*;

public class CheckBoxExample extends JPanel implements ActionListener,SwingConstants{
  static JFrame myFrame;
  protected JLabel label;
  JButton theButton;

  public CheckBoxExample(){
    Icon icon = new ImageIcon ("feet.gif");
    theButton = new JButton("My Feet",icon);
    add (theButton);
    add (new CheckBoxPanel(this));
  }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void actionPerformed(ActionEvent ae){
      String action = ae.getActionCommand();
      if (action.equals("north")){
        theButton.setVerticalTextPosition(TOP);
        theButton.setHorizontalTextPosition(CENTER);
      }
      else if (action.equals("south")){
        theButton.setVerticalTextPosition(BOTTOM);
        theButton.setHorizontalTextPosition(CENTER);
      }
      else if (action.equals("east")){
        theButton.setHorizontalTextPosition(RIGHT);
        theButton.setVerticalTextPosition(CENTER);
      }
      else if (action.equals("west")){
        theButton.setHorizontalTextPosition(LEFT);
        theButton.setVerticalTextPosition(CENTER);
      }
      else if (action.equals("center")){
        theButton.setHorizontalTextPosition(CENTER);
        theButton.setVerticalTextPosition(CENTER);
      }
    }
    public static void main(String args[]){
      myFrame = new JFrame("Checkbox Example");
      CheckBoxExample jt = new CheckBoxExample();
      myFrame.getContentPane().add("Center",jt);
      myFrame.setSize(400,250);
      myFrame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {System.exit(0);}
        });
      myFrame.setVisible(true);
    }
  }
```



图 23.15　每当用户选择不同的设置时，标签的对齐方式都发生变化

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

23.10.1 理解 CheckBoxPanel

看起来 CheckBoxPanel 的构造器与本书前面例子中的某些构造器很像。唯一真正的不同是该构造器还增加了从 JPanel(CheckBoxPanel)类中扩展出来的类。

23.10.2 改变对齐方式

正像用户从程序清单 23.7 里的 CheckBoxPanel 中回忆起来的那样,构造器要求传递一个 ActionListener 接口。用户应该注意到 CheckBoxExample 实现了 AcdionListener 接口。所以,当创建 CheckBoxPanel 时,用户自动而有效地将自己的类注册为接收来自复选框的 Action-Event 事件的监听器。

当然,用户现在感兴趣的是如何改变按钮上文本的对齐方式。ActionPerfomed( )方法首先获得动作命令,接着将动作命令与已知的每个结果比较,然后改变对齐方式。所以,如果命令是"north",那么用户应当只关心下面的代码:

```
if(action.equals("north")){
    theButton.setVerticalTextPosition(TOP);
    theButton.setHorizontalTextPositon(CENTER);
}
```

因为不知道此前哪一个被设置过,所以同时设置了水平位置和垂直位置。用户也可以按照 North-West 和 South-East 方式设置。

用户可能奇怪 TOP 变量和 CENTER 变量是从哪里冒出来的。在前面我们曾经提到过它们来自于 SwingConstants 接口。这个问题的答案非常简单:为了使用 SwingConstants,用户可以像在"增加图标"部分看到的那样实现该接口,也可以调用有资格的名字比如 SwingConstants.TOP。在本例中,CheckBoxExample 实现了 SwingConstants 接口,所以可以直接使用变量。

23.11 标签窗格

标签视足多个 GUI 设计的集成,它几乎是从 GUI 概念开始的。AWT 包含了一个名叫 CardLayout 的布局管理器,许多人用它来创建自己的标签窗口。但是 AWT 并不完全具备制作标签窗格这方面的能力。JFC 为了这个目的增加了 JTabbedPane 类。

JTabbedPane 自动处理创建标签窗格的图形事务。和 CardLayout 一样,它也允许用户在单击标签时隐藏和显示不同的页。图形标签最大的特点之一是它们可以被放置在四个标准位置中的任何一个上(顶、底、左或右)。程序清单 23.9 显示了如何在顶部创建两个标签窗格。

程序清单 23.9 JTabbedPane 提供

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;

public class TabExample extends JPanel {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    static JFrame myFrame;
    public TabExample(){
      JTabbedPane tabs = new JTabbedPane(SwingConstants.BOTTOM);
      Icon icon = new ImageIcon ("feet.gif");
      JButton button = new JButton(icon);
      JLabel label = new JLabel ("Hello World!");
      tabs.addTab("Hello World",label);
      tabs.addTab("Feet",icon,button);
      setLayou't(new BorderLayout());
      add(tabs,"Center");
    }

    public static void main(String args[]){
      myFrame = new JFrame("Tab Example");
      TabExample tabExample = new TabExample();
      myFrame.getContentPane().add("Center",tabExample);
      myFrame.setSize(400,200);
      myFrame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {System.exit(0);}
        });
      myFrame.setVisible(true);
    }
  }
```

23.11.1　理解 JTabbedPane

图 23.16 显示了运行 JTabbedPane 的结果。正如在其它例子中使用过的那样，main( )方法是很容易设计的。



图 23.16　变换标签得到不同的组件

在本例中，有两个标签窗格被加进了 tabs 对象，使用的是 addTabl()方法。在 addTab()上有三个变体，允许用户在标签上增加字符串和图标，下面是三种方法：

```
addTab(string title,Component component);
addTab(string title,Icon icon,Component component);
addTab(string title, Icon icon, Component component, String toolTip);
```

程序清单 23.9 显示了两个选项的用法：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
tabs.addTab("Hello world",label);
tabs.addTab("feet",icon,button);
```

component 对象是用作窗格的体。在本例中,一个标签上使用了按钮,在另一个上使用了普通的标签(Label)。总之,用户可以很容易地增加 JPanel 或其它的混合对象。

23.11.2 其它的 JTabbedPane 能力

在卡片布局中,用户可以使用插入顺序做为卡的顺序。与卡片式布局不同,使用 JTabbedPane 的用户也可以指定卡的顺序。这意味着当用户向系统中增加额外的标签窗格时,该标签窗格不一定是最后一个。为了做到这一点,用户可以使用 insertTab( )方法:

```
public void insertTab(String title, Icon icon, Component component, String tip, int index)
```

23.12 滑杆

滑杆类似于滚动条,允许用户在屏幕上拖动一个标志。典型的滑杆用于指定一个量。不幸的是在 AWT 中没有包含滑杆组件。但是 JFC 弥补了这个缺陷,它增加了 JSlider 类。

JSlider 可以显示主刻度、次刻度、主刻度和次刻度或什么都没有。这些刻度用于向用户显示状态。另外这些滑杆可以被垂直或水平地显示。程序清单 23.10 显示了几种垂直滑杆(如图 23.17 所示)。

程序清单 23.10 SliderExample.java——五个不同的滑杆

```
import com.sun.java.swing.*;
import com.sun.java.swing.border.*;
import com.sun.java.swing.event.*;
import java.awt.BorderLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class SliderExample extends JPanel{
  JLabel slider5Value;
  static JFrame myFrame;
  public SliderExample() {
    Box horizBox = Box.createHorizontalBox();
    JSlider slider1 = new JSlider (JSlider.VERTICAL, 0, 50, 25);
    slider1.setPaintTicks(true);
    slider1.setMajorTickSpacing(10);
    slider1.setMinorTickSpacing(2);
    slider1.setSnapToTicks(true);
    horizBox.add(slider1);
    horizBox.add(horizBox.createHorizontalStrut(15));

    JSlider slider2 = new JSlider (JSlider.VERTICAL, 0, 50,25);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    slider2.setPaintTicks(true);
    slider2.setMinorTickSpacing(5);
    horizBox.add(slider2);
    horizBox.add(horizBox.createHorizontalStrut(15));

    JSlider slider3 = new JSlider (JSlider.VERTICAL, 0, 50,25);
    slider3.setPaintTicks(true);
    slider3.setMajorTickSpacing(10);
    horizBox.add(slider3);
    horizBox.add(horizBox.createHorizontalStrut(15));

    JSlider slider4 = new JSlider (JSlider.VERTICAL, 0, 50,25);
    slider4.setBorder(LineBorder.createBlackLineBorder());
    horizBox.add(slider4);
    horizBox.add(horizBox.createHorizontalStrut(15));

    JSlider slider5 = new JSlider (JSlider.VERTICAL, 0, 50,25);
    slider5.setBorder(LineBorder.createBlackLineBorder());
    slider5.setMajorTickSpacing(10);
    slider5.setPaintLabels(true);
    horizBox.add(slider5);
    horizBox.add(horizBox.createHorizontalStrut(15));

    slider5Value = new JLabel("Slider5 value = 25");
    horizBox.add(slider5Value);

    slider5.addChangeListener(new ChangeListener(){
      public void stateChanged(ChangeEvent event){
        slider5Value.setText("Slider5 value = " + ((JSlider)event.getSource()).getValue());
      }
    });


    setLayout(new BorderLayout());
    add(horizBox,"Center");
  }
  public static void main(String args[]){
    myFrame = new JFrame("Slider Example");
    SliderExample sliderExample = new SliderExample();
    myFrame.getContentPane().add("Center",sliderExample);
    myFrame.setSize(300,300);
    myFrame.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent e){System.exit(0);}
    });
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
myFrame.setVisible(true);
```

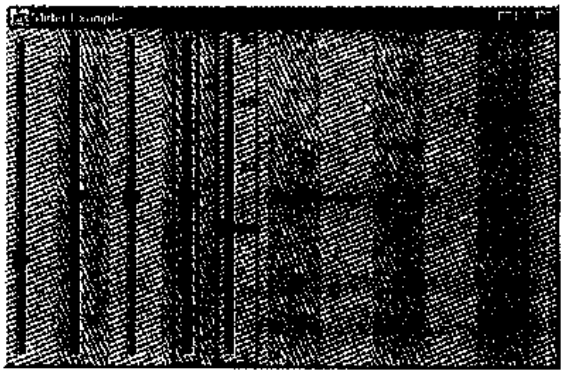


图 23.17 滑杆可以用来设置程序中的值

23.12.1 理解 SliderExample

SliderExample 使用了用户在 CheckBoxPamel 中看到的矩形面板。每个不同的滑杆被加进了水平的矩形面板。现在为了创建不同的滑杆，我们使用带有四个不同值的构造器，例如：

```
JSlider slider1 = new JSlider (JSlider.VERTICAL , 0 , 50 ,25);
```

构造器的第一个参数明显地是方向，它既可以是 SwingConstants.VERTICAL，也可以是 SwingConstants.HORIZONTAL。注意，因为 JSlider 实现了 SwingConstants 接口，所以我们才可以使用像 JSlider.VERTICAL 这样的值。

构造器的第二个参数和第三个参数分别是滑杆的最小值和最大值。最后一个参数用户可能已经猜出来了，它是滑杆的初始值。

23.12.2 设置刻度标志

缺省情况下滑杆不显示它的刻度标志(类似于滑杆 5)。但是，用户可以分别使用 SetMajorTickSpacing( )方法和 SetMinorTickSpacing( )方法独立地设置主刻度标志和次刻度标志。这两个方法都使用一个 int 型参数。该参数指定每个刻度之间的要素数。

23.12.3 捕捉滑杆的变化

为了捕捉滑杆中的变化，用户可以使用 JFC 集中的新事件：xxx.swing.event.ChangeEvent。ChangeEvent 事件发生在滑杆被移动时，所以在程序清单 23.10 中当 slider5 被移动时，最右边的标签发生了变化，该标签中包含有来自于该滑杆的值。

23.13 进度条

当某个活动要占用很长时间才能完成时，许多应用程序使用进度条来显示当前状态，并帮助用户知道该过程正在继续。AWT 中不包含进度条，JFC 填补了这个空白。JProgressBar 组

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

件是 JFC 为填补这一空白而开发的类。

程序清单 23.11 中的 ProgressBarExample 演示了如何使用 JProgressBar。它创建了一个与该进程并行的线程，并随着进程进展刷新滑杆，如图 23.18 所示。

程序清单 23.11　ProgressBarExample 使用了由线程控制的进度条

```
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class ProgressExample extends JPanel {
  ProgressThread progressThread;
  JProgressBar progressBar;
  static JFrame myFrame;

  public ProgressExample() {
    setLayout(new BorderLayout());
    progressBar = new JProgressBar();
    add(progressBar,"Center");

    JPanel buttonPanel = new JPanel();
    JButton startButton = new JButton("Start");
    buttonPanel.add(startButton);
    startButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
      startRunning();
      }
      });

    JButton stopButton = new JButton("Stop");
    buttonPanel.add(stopButton);
    stopButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
      stopRunning();
      }
      });
    add(buttonPanel, BorderLayout.SOUTH);
  }

  public void startRunning() {
    if(progressThread == null || !progressThread.isAlive()) {
      progressThread = new ProgressThread(progressBar);
      progressThread.start();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }
  }

  public void stopRunning() {
    progressThread.setStop(true);
  }

  public static void main(String args[]){
    myFrame = new JFrame("Hello World!");
    ProgressExample progressExample = new ProgressExample();
    myFrame.getContentPane().add("Center",progressExample);
    myFrame.setSize(200,100);
    myFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
      System.exit(0);}
      });
    myFrame.setVisible(true);
  }
}

class ProgressThread extends Thread {
  JProgressBar progressBar;
  boolean stopStatus = false;
  boolean aliveStatus = false;

  public ProgressThread(JProgressBar progressBar){
    this.progressBar = progressBar;
  }

  public void setStop(boolean value){
    stopStatus = value;
  }

  public void run () {
    int min = 0;
    int max = 50;
    progressBar.setMinimum(min);
    progressBar.setMaximum(max);
    progressBar.setValue(min);
    for (int x = min;x <= max;x++) {
      if(stopStatus){
        break;
      }else
        progressBar.setValue(x);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            try {
              Thread.sleep(100);
            } catch (InterruptedException e) {
              // Ignore Exceptions
            }
          }
        }
        aliveStatus = false;
      }
    }
```

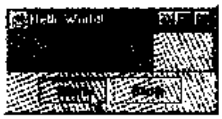


图 23.18　一旦 ProgrseeThread 被启动，进度条就开始一步步地穿过屏幕

23.13.1　理解 ProgressBarExample

为了演示有实际意义的进度条，用户还需要做一些实际的处理操作。实际使用的进度条不会像程序清单 23.11 中的 ProgressBarExample 那么复杂。当用户花时间消化了这个例子后，它就不再像现在看起来这么困难了。

23.13.2　创建和控制进度条

要想弄清楚本例中的进度条是如何工作的，用户需要看一下两个地方。首先，看一下进度条是如何被创建的，用户可以像看本章中其它例子那样先看一下构造器，可以看到下面这样两行语句：

```
progressBar = new JProgressBar( );
add(progressBar,"Cnter");
```

这两行与本章中其它的例子非常类似，没有什么出奇之处。第二个要看的地方是 ProgressThread 的 run( )方法，在这里用户可以找到设置和刷新进度条的所有调用。第一套方法是简单地设置进度条的初始状态。

```
progressBar.setMinimum(min);
progressBar.setMaximum(max);
progressBar.setValue(min);
```

如果用户回想一下第 19 章对滚动条的讨论就会发现，这些方法与那里的一些方法非常相似。这些方法的作用是设置进度条的最小值、最大值和初值。设置最小值和最大值的用处可能不是很明显，但是它隐含了刻度设置的一些因素。例如：如果用户希望显示读取文件的状态，并且希望从 10K 开始作标志，直到 5Mb，这时用户可以不使用 0 到 50 这种计数形式，而把最小值设为 10240，最大值设为 5242880。然后随着用户从文件中读取数据的进展，用户可以使用当前的位置刷新进度条；进度条负责处理所有的百分数计算。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

23.13.3 ProgressThread

在本例中额外的代码主要是用于处理刷新进度的线程。很明显该线程的主要用途是调用进度条的 setValue。

有一个警告需要说明。ProgressThread 调用了一个线程安全方法——setValue。在 JFC API 中大部分方法都不是线程安全的。这意味着,用户一般不能直接地从一个独立的线程调用这些方法。这样做会导致某些意外的用户不希望的结果。取而代之的是用户应该在 Swing 线程中完成大部分调用。

如果由于某种原因用户需要调用非线程安全的方法,用户可以使用 Swing 技术处理该线程。在 SwingUtilties 类中有两个方法:invokeAndWait( )和 invokeLater( )。这两个方法都要求传递 Runnable 对象(就像线程一样),但是它们将在 Swing 的事件线程中处理 run( )方法。这两个方法之间的差别是 invokeAndWait( )强迫 Runnable 对象立即被处理,并且用户的代码将阻滞在 invokeAndWait( )方法处,直到该方法调用完毕。invokeAndWait( )方法将排队等候 Runnable 对象,并在下一次机会里处理该对象,且与当前代码无关。通常用户将使用 invokeLater( )。

23.13.4 RrogressMonitor

因为显示代表某个运行过程和过程的状态的进度条是很平常的事,所以 JFC 提供了把这方面所有的东西结合在一起的类。这个类叫做 ProgressMonitor。

RrogressMonitor 实际上有一个智能的类,它能够在用户需要的时候做一些计算。如果任务在两秒内就能够完成。那么 RrogressMonitor 就不弹出来。用户可以设置在进度条弹出前用户需要等待的时间,但是一般地讲,两秒钟是很合适的时间。

程序清单 23.12 显示了如何将程序清单 23.11 中的 ProgreesBarExample 修改成为使用监视器的进度条程序。注意观察程序清单 23.12 和 23.11 的差别。用户可以在图 23.19 中看到程序清单 23.12 的运行结果。

```
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class ProgressMonitorExample extends JPanel {
  ProgressThread progressThread;
  static JFrame myFrame;

  public ProgressMonitorExample() {
    setLayout(new BorderLayout());
    JPanel buttonPanel = new JPanel();
    JButton startButton = new JButton("Start");
    buttonPanel.add(startButton);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        startButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                startRunning();
            }
        });

        JButton stopButton = new JButton("Stop");
        buttonPanel.add(stopButton);
        stopButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
            stopRunning();
            }
            });
        add(buttonPanel, BorderLayout.SOUTH);
    }

    public void startRunning() {
        if(progressThread == null|| ! progressThread.isAlive()) {
            progressThread = new ProgressThread(this);
            progressThread.start();
        }
    }

    public void stopRunning() {
        progressThread.setStop(true);
    }

    public static void main(String args[]){
        myFrame = new JFrame("Hello World!");
        ProgressMonitorExample progressMonitorExample = new
            ProgressMonitorExample();
        myFrame.getContentPane().add("Center",progressMonitorExample);
        myFrame.setSize(200,100);
        myFrame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
            System.exit(0);
            }
            });
        myFrame.setVisible(true);
    }
}

class ProgressThread extends Thread {
    ProgressMonitor monitor;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    boolean stopStatus = false;
    int min = 0;
    int max = 50;

    public ProgressThread(Component parent){
      monitor = new ProgressMonitor(parent,"Progress of Thread","Not Started",min,max);
    }

    public void setStop(boolean value){
      stopStatus = value;
    }

    public void run () {
      monitor.setNote("Started");
      for (int x = min;x <= max;x++) {
        if(stopStatus){
          monitor.close();
          break;
        }else{
          monitor.setProgress(x);
          monitor.setNote(""+(x*2)+"%");
          try {
            sleep(100);
          } catch (InterruptedException e) {
            // Ignore Exceptions
          }
        }
      }
    }
  }
```



图 23.19 ProgressMontor 产生一个子窗口中显示状态

理解 ProgressMonitorBar 与 ProgreesBarExample 程序不同，程序清单 23.12 不把 ProgressMonitorBar 的工作分在两个类中完成。'Start'和'Stop'按钮用于触发在 ProgressThread 对象中创建 ProgressMonitor。

ProgressThread 类 RrogressThread 类是本例中真正令人感兴趣的类。让我们从构造器开始，看 Monitor 对象是如何被建立的。

```
Public ProgressThread(Componcnt parent){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    monitor = new ProgressMonitor(parent, "Progress of Thread","Not Started", min, max);
}
```

ProgressMonitor 有一个构造器，它要求用户向它传递某些项目：

- 父窗口，像对话框或有模式 JDialog
- 窗口标签
- 初始化状态提示
- 滚动条的最小值和最大值

正如用户可以猜到的那样，用户可以在构造完 ProgressMonitor 以后改变这些值，其中只有一个不能修改：父窗口。这个值用于构造对话框，改变它并没有太大意义。

run( )方法　在本例中大部分工作是在 run( )方法中做的。当用户浏览 run()方法时，用户将会发现我们使用了比 ProgressExample 更多的方法。

首先，当用户在 ProgressExample 中设置 stopStatus 时，用户只是简单地结束了 run( )方法。但是在本例中，用户是通过监视器关闭了 run( )方法。通常，只要进度到达或超过最大值时，ProgressMonitor 将自动地关闭。但是，如果要在进度结束之前关闭它，就必须调用 close( )方法，当然，也可以通过把进度设为最大值来完成相同的任务。

```
If(stopStatus){
    monitor.close( );
    break;
}
```

其次，应该注意线程的刷新，用户为监视器刷新的不是一个值而是两个值。第一个刷新的是进度状态，与在程序清单 23.11 中做的一样，只是 ProgressMonitor 中用的方法是 setProgress( )，而 JProgressBar 中使用的是 setValue( )。

第二个方法更加有趣；它设置 ProgressMonitor 的提示。用户可以使用提示显示刷新信息，这种信息比进度条显示的信息更直观。如果用户要读一堆文件，那么用户可能希望显示正在工作的当前文件。

23.13.5　ProgressMonitorInputStream

进度条另一个常见的用法是显示读取文件的进展，比如 FileInputStream.。像 ProgressMonitor 一样，为了方便用户编程，JFC 为从流中读取信息增加了一个监视器，该新类叫做 ProgressMonitorInputStream，它与 ProgressMonitor 一样，也创建它自己的对话框窗口。

因为 ProgressMonitorInputStream 类似 ProgressMonitor，所以用户基本上可以使用程序清单 23.12 中的类，只要用程序清单 23.13 中 ProgtessThread 类做一下替换就可以看出 ProgressMonitorInputStream 是如何工作的。

程序清单 23.13　通过用 ProgressMonitorInputStream 替换 ProgressMonitor 来修改 ProgressThread

```
class ProgressThread extends Thread {
    boolean stopStatus = false;
    BufferedInputStream in;

    public ProgressThread(Component parent,String fileName){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    try{
      in = new BufferedInputStream(
           new ProgressMonitorInputStream(
               parent,
                 eading * + fileName,
               new FileInputStream(fileName)
               )
           );
    }catch (FileNotFoundException exception){
      System.out.println("File not found:" + fileName);
    }
  }

  public void setStop(boolean value){
    stopStatus = value;
  }

  public void run () {
    int readVal;
    try{
      while (! stopStatus && ((readVal = in.read())! =-1)){
        //if this was for real, you'd do something here
      }
    }catch (IOException ioe){
      System.out.println("Exception while reading");
    }
    System.out.println("done");
  }
}
```

理解 ProgressMonitorInputStream 程序清单 23.13 中真正的工作来自于该类的构造器。在我们深入研究它之前，我们应该先注意一下 ProgressMonitorInputStream 是从 InputStream 扩展来的，并且 ProgressMonitorInputStream 一般也可以用作自定义的输入流，像 BufferInputStream 或 ObjetInputSteam 一样。

ProgressThread 的构造器基于 FileInputStream 创建了 ProgressMonitorInputStream。在功能方面，它涵盖了 FileInputStream。所以显得与 ProgressMonitorInputStream 没有什么不同，它被封装在 ProgressMonitorInputStream 中了。

```
in = new BufferedInputStream(new ProgressMoniterInputSteram(parent,
"Reading" + fileName, new FileInputStream(fileName)));
```

ProgressMonitorInputStream 的构造器被包含在语句行中了，它要求三个组件：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 父容器，这和 ProgressMonitor 一样
- 监视器信息
- InputStream 封装，它提供数据源

与 ProgressMonitor 不同，ProgressMonitorInputStream 不允许用户直接修改显示的信息。如果用户需要修改它，用户可以使用 getProgressMonitor( )方法请求 ProgressMonitor。

run( )方法　ProgressThread 的 run( )方法只是简单地从流中读取信息直到结束。为了使本例更加生动一些，需要用户编写 while( )循环体。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 24 章　高级 JFC

本章内容：

24.1　模型 – 视 – 控制——JFC 设计
24.2　菜单和工具条
24.3　列表框和组合框
24.4　使用表格
24.5　树
24.6　使用 JEditor 显示 HTML 文件
24.7　创建图标
24.8　JFC 小应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

24.1　模型 – 视 – 控制——JFC 设计

JFC 整个系统是基于模型 – 视 – 控制概念设计的，模型视控制在英文中简称 MVC。MVC 系统把每个接口组件分成三个部分，如图 24.1 所示。

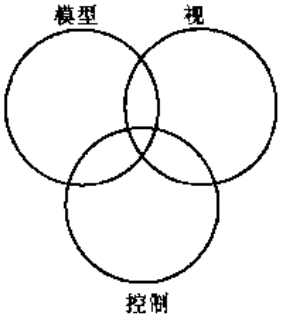


图 24.1　模型视控制设计把接口分成三个部分

这种设计在过去曾被其它的系统成功地应用过，但在 Java 中还是第一次应用于 JFC。每个这种组件都有它自己的请求和责任集。下面是 MVC 的三个部分：

模型：模型负责了解和维护组件的状态。例如，按钮的模型必须知道按钮是否被按下，或者是否被启用。在切换方面，当按钮被按下时，模型负责将这个变化通知所有的事件监听器。

视：视负责组件的显示。它主要涉及 paint( )方法，但这也意味着要精确地知道什么东西需要被绘制。所以如果组件当鼠标放在它上面时有不同的外观，那么视必须能够显示正常的视和当鼠标在组件上时的视。

控制：控制负责管理由组件接收到的事件。这些事件既可以是来自用户的(比如鼠标单击)，也可以是来自系统的(比如计时器事件)。控制必须断定用该事件做什么和通知模型、视和任何其它的监听器。

因为视和控制之间的通信是非常复杂的，所以大部分 JFC 把它们结合成一个单独的叫做“代表”(delegate)的组件，产生一个类似于图 24.2 所示的设计模式。

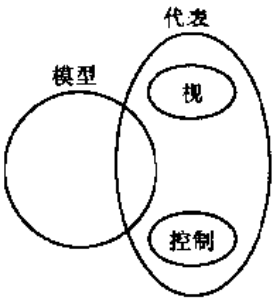


图 24.2　通过合并视和控制 JFC 简化设计

24.1.1　比较 MVC 和 AWT 设计

乍一看，用户可能认为 MVC 是基本上针对 Java 的，实际上，AWT 设计也确实有许多这方面的思想。在 AWT 里，组件有一个名叫对等的对象。对等实际是本地的组件，例如，如果用户在 Windows 下运行 Java 程序并创建一个 java.awt.Button 对象，则按钮从 Windows 上获得按钮对等，它实际上是本地 Wnidows API 按钮。对等有效地拥有了所有 JFC 的代表拥有的相同特性。AWT 模式和 MVC 系统的工作方式最基本的不同是对等设计使得改变按钮工作方

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

式或外观变得很难。总之用户将依赖于该系统(Windows)给出的按钮。

另一个重要的差别是在 AWT 设计中,MVC 的模式部分被嵌入在组件本身中了。所以出于某种原因,如果用户想要改变模式的工作方式,用户必须检查所有涉及到的处理,使得新的模式紧密地与源模式合在一起。

24.1.2 这种模式的组件用在哪里

可能理解 JFC 的 MVC 设计最易混乱的东西莫过于断定该组件适合什么位置了。例如,如果用户观察 JButton,它就自己在做工作吗? 它不是模型、视,也不是控制者;JBatton 是组件。看一下图 24.3 所示的两个图。左边的图类似于图 24.2,只有一点不同是 MVC 被显示为生存在组件内部。在图 24.3 右边的部分用户可以看到对象简图,它显示了模型、视和控制全部生存在一个"有"关系的组件中。

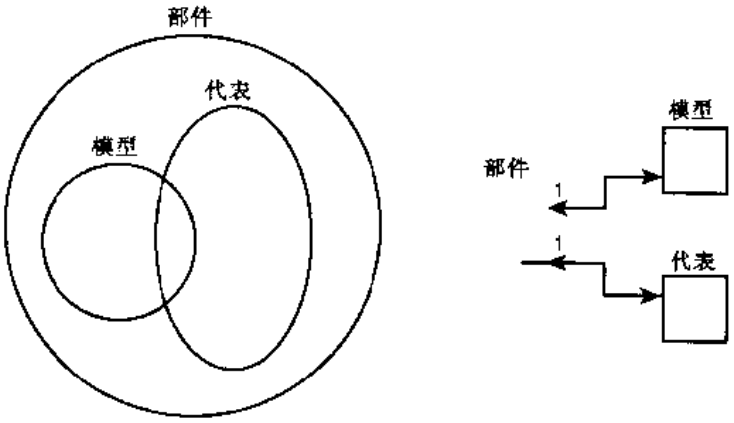


图 24.3 组件"有一个"模型和代表(视和控制者)

24.1.3 深入研究 MVC 组件是如何工作的

为了更好地理解这三个部分(组件、模型和代表)是如何在一起工作的,让我们看一个示例。用户可能奇怪这里的项目如何实际地结合在一起工作,所以我们选择两个示例来看它们实际是如何工作的。

在屏幕上绘制 如果用户是一个有经验的 Java 程序开发者,并且创建了一些有兴趣的小应用程序,或者用户阅读了前面涉及 AWT 的章节,用户会记得 java.awt.Component 有一个名为 paint( )的方法。每当组件需要在屏幕上显示时就要调用这个方法。事实上,如果用户回过头来看一下第 23 章"JFC—Java 基础类"中第一个介绍如何使用 JFC 的例子(程序清单 23.1)就会发现,paint( )方法用于显示 hello world;同样在小应用程序"Hello World"(如程序清单 14.5 所示)中它也是如此。

那么 Paint( )方法如何为 JFC 组件如 JButton 工作呢? 用户刚刚了解到:视是负责绘制按钮的。用户也知道视是按钮的"部分",但它实际不是按钮。所以用户可能奇怪这里的工作机制。总之,用户可以对 JButton 调用 add( ),而不是 JButton 的视。绘制线程将绘制被增加的组件,但是不知道关于代表对象的任何信息。

当 Java 系统的任何组件需要在屏幕上显示时,或者通知给 update( )方法时,屏幕重画线程将对该对象调用 paint( )方法。AWT 组件就是这样做的,但 JFC 组件不是这样的(注意:JFC 的 JComponent 不是从 AWT 的 Component 类扩展来的)。在 JFC 中,JComponent 负责确

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

保对所有组件的绘制都被正确地处理，包括所有子组件。当它绘制自己时，paint( )方法将 Graphics 对象传给视并指示视绘制。比较有效地，用户可以认为 JComponent 的 Paint( )方法像下面的代码那样工作：

```
pablic void paint (Graphis g) {
        ui.paint(g);
}
```

虽然实际的 JComponent 的 paint( )方法非常复杂，因为它要求绘制所有的子对象并容纳在 JFC 分层的视中，并且理解视的适用方式也很困难，但是它确确实实工作得很完美。

改变值　现在用户可能理解了 paint()方法如何利用视绘制屏幕显示内容，但是仍然不知道如何改变值。例如，如果某个按钮被按下，那么信息是如何分层传送的呢？

该任务的第一部分首先是用控制器接收事件。这个工作通过标准的 java.awt.events，比如 MouseListener 和 MouseMotionListener 来完成的。正如用户早些时候看到的那样，控制器是典型地被捆绑在视上的。所以当事件发生时是由视（更确切地说是视的内置类）来接收 mousePressed( )事件的。

在 mousePressed( )方法内部，控制器通知模型状态已被修改，由它来执行必要的修改，然后可能需要告诉该组件要做某件事情。

让我们看一下 JButton 如何捆绑这个序列。当鼠标事件发生时，事件被捆绑在控制器上。对于基本的用户界面（UI），这个类叫做 BasicButtonListener。在 mousePressed( )方法内，控制器执行两个任务。首先，它告诉模型将“被按下”状态设置为 true。然后它告诉组件（JButton）请求聚焦。注意该模型并不实际地直接去捆绑事件，而是被告知来自控制器的适当修改。

当模型收到来自于控制器的修改命令时，模型负责设置“被按下”状态，然后告诉视它需要被重画（以反映新的修改）。因为是由模型负责调用 repaint( )方法的，所以用户可以多方面地修改按钮的状态，并且在任何情况下都会被正确地刷新。例如，用户可以通过按下按钮来改变按钮的状态（像用户刚刚看到的那样），或者用户可以通过调用 JButton 的 setSelected( )方法或 doClick( )方法来改变按钮的状态。在这几种情况下，因为模型刷新了视，所以状态的改变会出现在屏幕上。

获取值信息　最后，用户应当考察一下当用户决定请求来自组件的信息时会发生什么情况。正如用户已经看到的那样，模型是负责提供信息的。所以当用户询问组件信息时，用户实际上是在询问模型。

让我们再看一下 JButton。如果用户跟着前面部分设置的方案执行下来的话，将会回忆起已经在按钮上按下了鼠标键。现在我们希望知道 JButton 状态。

正常地，用户将会获得由 JButton 的 isSelected( )方法查询的按钮状态。现在，正如用户可能已经猜到的那样，如果查看一下 JButton 类的 isSelected( )方法的内部就会发现，它实际上是从模型那里请求该状态并返回该值的。

24.1.4　为什么要理解 MVC

在许多应用程序中，用户可能根本就不需要知道模型－视－控制（MVC）模型是如何工作的。事实上，在第 23 章中用户学习 JFC 的时候就根本不知道该模型和该代表做了什么事情。这种情况不是不常见的。但是对于复杂的组件。比如树和表，理解 MVC 则是至关重要的；树

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

和表将在本章后面的部分中介绍。

24.1.5 使用 JFC 的可插入的外观 – 和 – 感觉

除了能够改变该系统的单个组件以外，JFC 的模型 – 视 – 控制设计还有一个真正迷人的副作用。这个副作用允许 JFC 设计变异组件的外观和无缝地修改对某些东西的感觉(反应)。这意味着用户不必改变代码行就可以改变应用程序的外观，甚至在运行时刻也可以改变。这个完整的概念叫做"可插入的外观 – 和 – 感觉"图 24.4、24.5 和 24.6 显示了某个应用程序的三种完全不同的视：Java 视、Windows 视和 Motif 视。

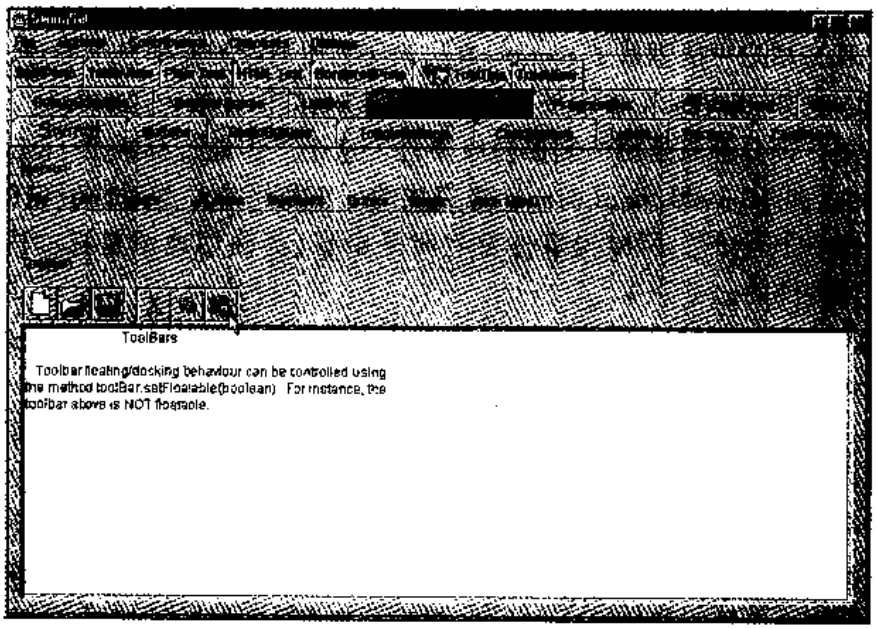


图 24.4 使用 Java 的外观 – 和 – 感觉再现的应用程序

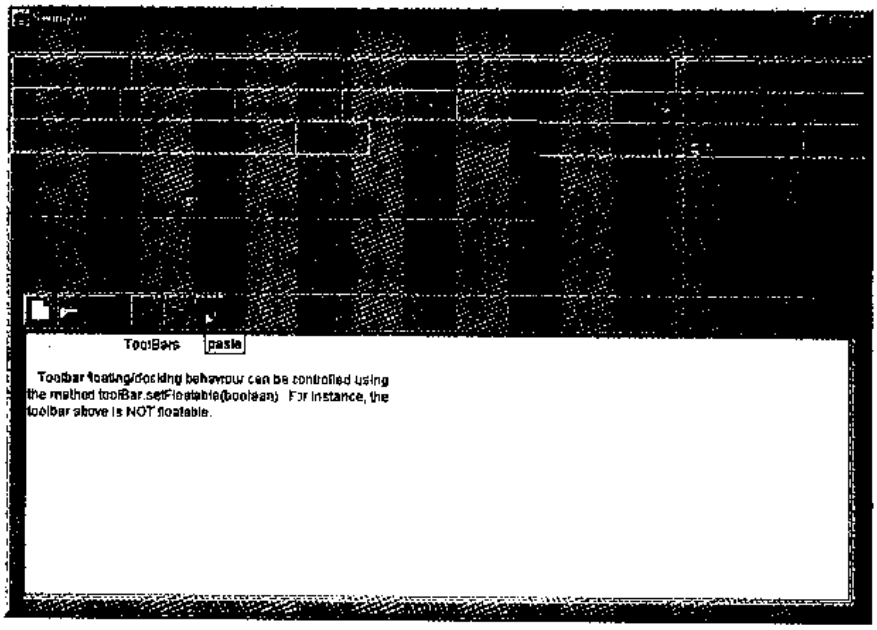


图 24.5 使用 Windows 的外观 – 和 – 感觉再现的应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

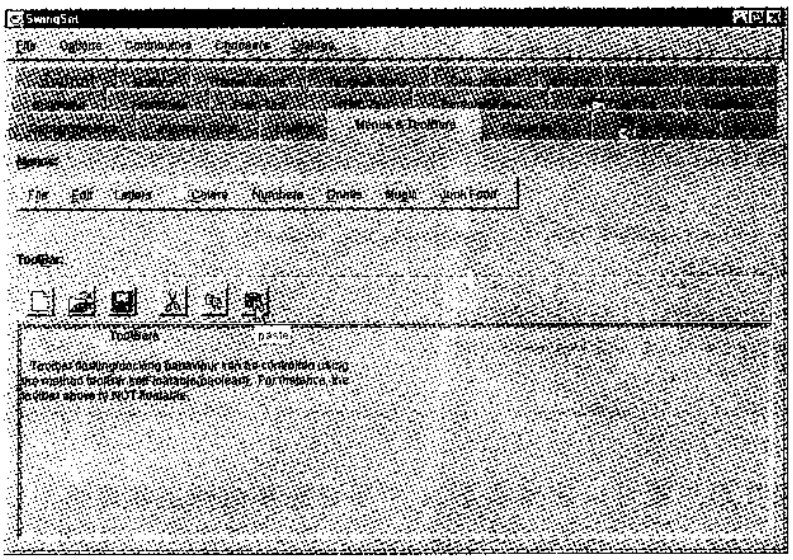


图 24.6　使用 Motif 的外观 - 和 - 感觉再现的应用程序

24.1.6　修改外观 - 和 - 感觉

修改应用程序的外观 - 和 - 感觉是非常容易的。要想改变它，用户只需要知道一件事情：负责为该外观 - 和 - 感觉提供各种用户界面组件的 Look - and - feel 类的名字。用户可以认为该 look - and - feel 类是该外观 - 和 - 感觉的驱动程序。各种外观 - 和 - 感觉都被捆绑在 JFC 上，它们的 look - and - feel 类被列在表 24.1 中。

表 24.1　JFC 载有的各种外观-和-感觉驱动程序

外观 - 和 - 感觉名	类　　名
基本外观和感觉——大部分外观 - 和 - 感觉的基本内核	com.sun.java.swing.plaf.basic.BasicLookAndFeel
Java 外观 - 和 - 感觉——新的很酷的外观 - 和 - 感觉。在本章和第 23 章中的大部分实例都使用了 Java 外观 - 和 - 感觉	com.sun.java.swing.plaf.metal.MetalLookAndfeel
Macintosh 外观 - 和 - 感觉——汇集了 Macintosh 工作方式的外观 - 和 - 感觉　注意：由于商标的原因，缺省情况下在除 Macintosh 以外的其它任何平台上不能使用这种外观 - 和 - 感觉	com.sun.java.swing.plaf.mac.MacLookAndFeel
Motif 外观和感觉——汇集了许多图形化的 UNIX 计算机的视的外观 - 和 - 感觉	com.sun.java.swing.plaf.motif.MotifLookAndFeel
复合的外观 - 和 - 感觉——允许同时有多种用户界面与某个组件关联的外观 - 和 - 感觉	com.sun.java.swing.plaf.multi.MultiLood AndFeel
Windows 外观 - 和 - 感觉——模拟 Windows 95/NT 的外观 - 和 - 感觉；由于商标的原因，缺省情况下在其它的非 Windows 平台上不能使用这种外观 - 和 - 感觉	com.sun.java.swing.plaf.windows.Windows

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

当用户在表 24.1 中找到了需要的外观 - 和 - 感觉以后,用户就可以着手改变用户的应用程序了。程序清单 24.1 显示了使用 Windows 外观和感觉的方法。

```
try{
  UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.
WindowsLookAndFeel");
} catch (java.lang.ClassNotFoundException e){
  System.out.pritln("Look and feel not found");
}
```

很明显,现在用户就可以使用表 24.1 中用户想要的外观和感觉来替换程序中的"com.sun.java.swing.plaf.windows.WindowsLookAndFeel"字符串。从技术上讲,这就是用户需要做的事情。但是,如果用户想要在程序运行中改变外观和感觉。即用户希望在运行时刻支持不只一种外观和感觉),那么用户也需要通知已经可视的组件重新刷新。用户可以用两种办法做这件事情。第一种办法是人工地在程序中每个组件上调用 updateUI( )方法。updateUI( )方法强迫该组件重新获得它的代表并重绘自己。很明显,如果用户在系统中有许多组件需要处理时要做大量的工作,它需要创建大量的耦合连接。SwingUtilities 类提供的方法可以使得这项任务变得非常简单。但是在用户可以使用 swing 实用程序之前,用户必须知道在所有其它视的根上的组件。典型地,该组件是 JFrame 或程序启动的 JApplet。当用户知道这个情况后,用户就可以调用下面的方法了:

```
SwingUltilities.updateComponentTreeUI(rootComponent);
```

在接下来的部分里,程序清单 24.2 将把这些东西全部放在一个完整的示例中。

24.2 菜单和工具条

在许多系统中菜单和工具条是普通的用户界面组件。AWT 提供了菜单,但未提供工具条。JFC 则既提供了菜单也提供了工具条。相应的 JFC 类是 JMmenuBar 和 JToolBar。JMenuBar 和 JToolBar 都是标准的 JComponents。用户可以把它们放在任何想放的位置上,不必局限在标准位置上。如果用户希望菜单条位于文本域下方,那么用户就可以这么做。

用户可以创建一个像程序清单 24.2 所示的例子。它允许用户改变 JTabbed 窗格的外观和感觉,如图 24.7 所示。在该例中增加了菜单条,该菜单条拥有 File 菜单和 L&F 菜单。File 菜单实际上什么事情也做不了,只是为了好看。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
public class MenuBarExample extends JPanel implements ItemListener{
  static JFrame myFrame;
    Font myFont = new Font("ialog", Font.PLAIN, 12);

  public MenuBarExample(){
//  setFont(myFont);
    JTabbedPane tabs = new JTabbedPane(SwingConstants.BOTTOM);
    Icon icon = new ImageIcon ("feet.gif");
    JButton button = new JButton(icon);
    JLabel label = new JLabel ("Hello World!");
    tabs.addTab("Hello World",label);
    tabs.addTab("Feet",icon,button);
    setLayout(new BorderLayout());
    add(tabs,"Center");
    add(createMenu(),"North");
  }

  public JMenuBar createMenu(){
    JMenuBar menuBar = new JMenuBar();
    // File Menu - create this so we have at least two menu options
    JMenu file = (JMenu) menuBar.add(new JMenu("File"));
    file.setMnemonic('F');
    JMenuItem mi; //Temporary place holder
    //Add several items under 'File' these won't do anything.
    mi = (JMenuItem) file.add(new JMenuItem("Open"));
    mi.setMnemonic('O');
    mi = (JMenuItem) file.add(new JMenuItem("Save"));
    mi.setMnemonic('S');
    mi = (JMenuItem) file.add(new JMenuItem("Save As..."));
    mi.setMnemonic('A');
    file.add(new JSeparator());
    mi = (JMenuItem) file.add(new JMenuItem("Exit"));

    // Look and Feel Menu
    JMenu options = (JMenu) menuBar.add(new JMenu("L&F"));
    options.setMnemonic('L');

    // Look and Feel Radio control
    ButtonGroup group = new ButtonGroup();
    mi = options.add(new JRadioButtonMenuItem("Windows Style Look and Feel"));
    mi.setActionCommand("java.swing.plaf.windows.WindowsLookAndFeel");
    //If the current look and feel is windows, select this item.
    mi.setSelected(UIManager.getLookAndFeel().getName().equals("Windows"));
    group.add(mi);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        mi.addItemListener(this);
        //  mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_1, ActionEvent.ALT_MASK));

        mi = options.add(new JRadioButtonMenuItem("Motif Look and Feel"));
        mi.setActionCommand("java.swing.plaf.motif.MotifLookAndFeel");
        mi.setSelected(UIManager.getLookAndFeel().getName().equals("CDE/Motif"));
        group.add(mi);
        mi.addItemListener(this);
        //  mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_2, ActionEvent.ALT_MASK));

        mi = options.add(new JRadioButtonMenuItem("Metal Look and Feel"));
        mi.setActionCommand("java.swing.plaf.metal.MetalLookAndFeel");
        mi.setSelected(UIManager.getLookAndFeel().getName().equals("Metal"));
        //  metalMenuItem.setSelected(true);
        group.add(mi);
        mi.addItemListener(this);
        //  metalMenuItem.setAccelerator(
➡KeyStroke.getKeyStroke(KeyEvent.VK_3, ActionEvent.ALT_MASK));
        return menuBar;
    }

    public void itemStateChanged(ItemEvent e) {
        Component root = myFrame;
        //Bump the cursor into a wait mode while we make this change
        root.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
        //Get the source of the event.
        JRadioButtonMenuItem button = (JRadioButtonMenuItem) e.getSource();
        try {
            if(button.isSelected()) {
                UIManager.setLookAndFeel(button.getActionCommand());
                button.setEnabled(true);
                SwingUtilities.updateComponentTreeUI(myFrame);
            }
        } catch (UnsupportedLookAndFeelException exc) {
            // Error - unsupported L&F
            button.setEnabled(false);
            System.err.println("Unsupported LookAndFeel: " + button.getText());
        }catch (Exception exc2){
                System.err.println("Couldn't load Look and feel" + button.getText());
        }

        root.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public static void main(String args[]){
      myFrame = new JFrame("MenuBar Example");
      MenuBarExample menuExample = new MenuBarExample();
      myFrame.getContentPane().add("Center",menuExample);
      myFrame.setSize(400,200);
      myFrame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {System.exit(0);}
        });
      myFrame.setVisible(true);
    }
}
```

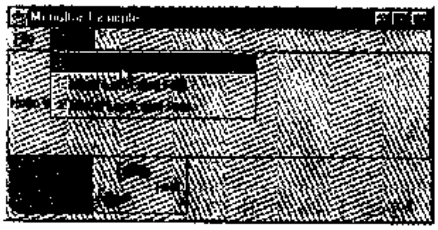


图 24.7　使用菜单选项可以改变外观和感觉

24.2.1　理解 MenuBar 示例

在这个例子中需要仔细研究的方法有两个：createMenu( )方法和 itemStateChanged( )事件方法。createMenu( )方法产生 JmenuBar，稍后 JMenuBar 将被放置在 JPanel 的“北”部。

createMenu( )　要想理解 createMenu( )方法，最重要的是理解 JMenuBar 基础。JMenuBar 实际上只是水平方向的容器，它包含有菜单，典型的菜单有 File 和 Edit。菜单本身也是容器，它包含有 MenuItems。MenuItems 才是可供人们实际选择的项目，例如 Open 或 Save。当然，如果用户已经使用 AWT 建造过菜单，那么这些都是老套路了。

创建菜单　创建菜单的第一个任务是创建 MenuBar。该菜单条将拥有两个菜单，File 和 L&F。在菜单条上用户需要分别为每个菜单增加 JMenus。当用户开始增加 MenuItems 时有趣的事情就来了。JMenuBar 的工作方式非常类似于 JButton。它可以拥有 Mnemonics、ActionEvents 和 ItemEvents。看一下 createMenu( )的头几行，第一个菜单(File)和该菜单的第一个菜单项(Open)就是在这里创建并加进去的。

```
JMenu file = (JMenu) menuBar.add(new JMenu("File"));
File.setMnemonic('F');
JMenuItem mi; //临时的位置占有者
mi = (JMenuItem)file.add(newJMenuItem("Open"));
```

除了菜单项以外，用户可能希望在项目之间增加分隔，比如在 Save As 选项和 Exit 选项之间：

```
file.add(new JSeparator( ) );
```

增加复选框菜单按钮　在该方法的后半部分，用户开始增加 L&F 菜单。在本例中，用户

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

不要使用常规的 MenuItem 类。因为用户只希望有一个同时只有一个被选中的选项，还因为希望有一个漂亮的可视的用于显示某个选项被选中的暗示，所以用户需要使用 JCheckBoxMenuItem。该类是从普通的 JMenuItem 扩展来的，它增加了显示复选框和标签的能力。第一个选择项目的代码是：

```
ButtonGroup group = new ButtonGroup( );
mi = Inf.add(new JCheckBoxMenuItem("Windows Style Look and Feel") );
mi.setActionCommand("Java.swing.plaf.windwows.WindowsLookAndFeel");
//如果当前的外观和感觉是 Windows 外观和感觉，那么就选择该项。
mi.setSelected(UIManager.getLookAndFeel( ).getName( ).equals("Windows") );

group.add(mi);
```

注意在这里使用了 ButtonGroup 类，它可以保证每次只能有一个选项被选中，所有的复选框菜单项都加在该组中。

24.2.2 加速键

用户已经看到过如何在 JFC 组件上使用助记符。程序清单 24.2 中的每个菜单项都有自己的助记符。但是只有当该组件可视时才可以使用助记符。在本章的 MenuItem 中使用助记符是很方便的，只要弹出式菜单被显示出来，程序的使用者就可以使用热键去选择选项了。但是如果用户有时希望使用组合键该怎么办？

因为弹出式菜单平时是不可见的，所以助记符是不起作用的。但是用户可以使用另一个技巧允许用户不管菜单是否可见都可以按组合键。这种技术叫做加速键。加速键捕捉未被其它设备使用但被按下的任何键（这意味着如果有一个活动的助记符与加速键相冲突，那么助记符优先发挥作用）。

setAccelerater( )方法被用于设置加速键。setAccelerater( )方法要求 KeyStroke 对象作参数。KeyStroke 是 JFC 中的另一个新类，它被用于表示组合键，包含一个字符和一个键掩码，比如 Shift 或 Alt。所以用户可以支持像 Ctrl + b 这样的组合键。掩码来自于 awt.event.ActionEvent 类。正如用户可以回忆起来的那样，它们是位掩码，包括下列值：

- java.awt.AceionEvent.SHIFT _ MASK
- java.awt.ActionEvent.CTRL _ MASK
- java.awt.ActionEvent.META _ MASK
- java.awt.ActionEvent.ALT _ MASK

所以，如果用户希望捕捉 Alt + 1 组合键，那么用户可以对该菜单项使用下列语句：

```
mi.setAccelerator(Keystroke.getKeyStroke(keyEvent.VK _ 1,ActionEvent.ALT _ MASK));
```

因为掩码是位标志，所以用户可以使用按位与（&）来组合它们。例如，如果用户想使用组合键 Shift + Ctrl + B，那么用户就可以使用下面的语句把两个掩码组合起来：

```
mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK-1,
        actionEvent.CTRL _ MASK&ActionEvent.SHIFT _ MASK) );
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

24.3　列表框和组合框

列表框可能是用途最单一的用户界面组件之一，它用于显示几个项目。组合框是列表框的另一种变体。AWT 包含了列表框和组合框(由 Choice 类描述)，但是它们都只限于显示字符串。JFC 增加了 JList，它是完全不同的列表框类型。JCombBox 与 AWT 的 Choice 有几方面的不同，它实际上使用的是 JList 类的弹出式显示。除了显示字符串以外，JList 还可以显示各种类型的对象，不仅可以显示字符串，而且可以显示有关联的图标。再则，列表框的视可以基于大量的条件变化。所以，如果用户希望当项目被选中后产生不同的视时，使用 JList 和组合框很容易做到。首先；用户需要建立简单的列表框和组合框。通常用户希望有一个来自固定列表的列表框。程序清单 24.3 显示了一个列表框和一个组合框，它们都有固定的选项集(见图 24.8)。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.wing.border.*;

public class ListComboExample extends JPanel{
  static JFrame myFrame;

  String values[] = {"Joe","Shawn","Gabe","Jim","Bill","Jeremy"};
  public ListComboExample(){
    setLayout(new GridBagLayout());
    JList list = new JList(values);
    list.setVisibleRowCount(4);
    JScrollPane pane = new JScrollPane();
    pane.setViewportView(list);
    add(pane);

    JComboBox combobox = new JComboBox(values);
    add(combobox);
  }

  public static void main(String args[]){
    myFrame = new JFrame("List and ComboBox Example");
    ListComboExample jt = new ListComboExample();
    myFrame.getContentPane().add("Center",jt);
    myFrame.setSize(400,250);
    myFrame.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent e){System.exit(0);}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        });
        myFrame.setVisible(true);
    }
}
```

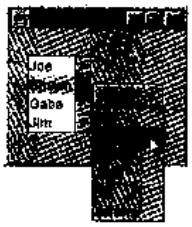


图 24.8 排在一起的 JList 和 JCombBox

24.3.1 理解 ListCombExample

现在,程序清单 24.3 的内容对于用户来说都是很直观的了。该列表框最有意思的部分是 JScrollPane 的使用。JList 并没有自己实现滚动。这是一个很有趣的设计,它允许用户不必捆绑标准的滚动就可以做许多有趣的事情。在大部分情况下,用户可以简单地创建 JScrollPane 并将它的视点设置在列表框上。注意它是窗格,不是一个列表框,实际上是在向面板上增加。

24.3.2 列表视模型

虽然从缺省的列表中创建列表框是很有用的,但是动态列表框更有趣。正如用户可以从在本章前面对模型-视-控制的讨论中回想起来的那样,由模型负责为组件提供数据。所以要想使用列表中的数据,必须使它能够感觉到用户对模型的修改。列表框使用基本的列表调用 ListModel.。ListModel 是一个包含有四个方法的接口,可以很容易扩展 AbstractListModel 类,除此之外它只定义了两个方法:getElementAT( )和 getSize( )。

组合框的模型稍微复杂些。首先,它是从 ListModel 扩展来的,这意味着用户可以基于一个已有的 ListModel 创建 CombBoxModel,但是在此之后,用户必须定义两个额外的方法:setSelectedItem( )和 getSelectedItem( )。程序清单 24.4 演示了如何通过修改程序清单 24.3 使用模型替换固定数据列表。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.border.*;

public class ListComboExample2 extends JPanel{
  static JFrame myFrame;

  public ListComboExample2(){
    setLayout(new GridLayout(2,2));
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    JList list = new JList(new ListModelExample());
    list.setVisibleRowCount(4);
    JScrollPane pane = new JScrollPane();
    pane.setViewportView(list);
    add(pane);

    JComboBox combobox = new JComboBox(new ComboModelExample());
    add(combobox);

  public static void main(String args[]){
    myFrame = new JFrame("List and ComboBox Example");
    ListComboExample2 jt = new ListComboExample2();
    myFrame.getContentPane().add("Center",jt);
    myFrame.setSize(400,250);
    myFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {System.exit(0);}
      });
    myFrame.setVisible(true);
  }

  class ListModelExample extends AbstractListModel{
    String values[] = {"Joe","Shawn","Gabe","Jim","Bill","Jeremy"};
    public Object getElementAt(int index){
      return values[index];
    }

    public int getSize(){
      return values.length;
    }
  }

  class ComboModelExample extends ListModelExample implements ComboBoxModel{
    Object item;
    public void setSelectedItem(Object anItem){
      item = anItem;
    }
    public Object getSelectedItem(){
      return item;
    }
  }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

24.4 使用表格

有些人争论这样一个观点:电子表格的发明是引起计算机趋于流行的一项革新。电子表格的主要特性之一是能够在表格中显示二维数据。虽然不是羽翼丰满的电子表格,但是 JFC 拥有一个非常可扩展的表格组件叫做 JTable。

用户可以从查看一个最简单的 JTable 版本开始了解 JTable,该版本定义了一个静态的数据集(如图 24.9 所示)。程序清单 24.5 显示的就是这个示例,在该表格中列出了本书的前几章目录。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.border.*;

public class TableExample extends JPanel{
  static JFrame myFrame;
  String data[][] = {{"1","Introduction"},
    {"2","What Java can Do for You"},{"3","JAVA Design"},{"4","Installing JAVA"},
    {"5","JDK tools"},{"6","Object-Oriented Programming"},{"7","Hello world"},
    {"8","Data Types"},{"9","Methods"},{"10","Using Expressions"}};
  String columnNames[] = {"Chapter Number","Chapter Title"};

  public TableExample(){
    setLayout(new BorderLayout());
    JTable table = new JTable(data,columnNames);
    JScrollPane pane = JTable.createScrollPaneForTable(table);
    add(pane);
  }
  public static void main(String args[]){
    myFrame = new JFrame("Table Example");
    TableExample tableExample = new TableExample();
    myFrame.getContentPane().add("Center",tableExample);
    myFrame.setSize(400,250);
    myFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {System.exit(0);}
      });
    myFrame.setVisible(true);
  }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

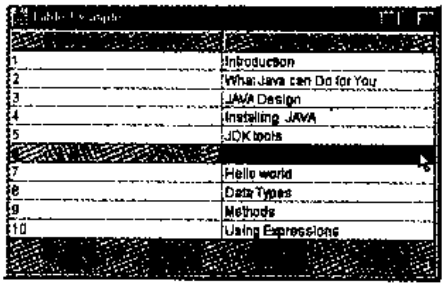


图 24.9　JTable 漂亮地显示了二维数据

24.4.1　理解 TableExample

TableExample 使用了 Table 最简单的构造器。该构造器使用一个二维数据和一个列头标签集作为参数。在 TableExample 中，用户看到的是代表数据和列头的字符串数组。数据不仅限于字符串，在本例中使用字符串是最容易处理的。用户也可以使用其它各种组件。

像 JList 一样 Table 也没有自己实现滚动。但与 JList 不同，Table 有一个很方便的方法可以创建滚动窗格并可以把表格和表格头放在适当的位置上。和 JList 一样，Table 也是一个窗格，它本身并不是表格，当加到面板上时它就成了表格了。

24.4.2　表格模式

虽然完整地介绍所有的 Table 选项超过了本书的范围，但是如果不使用模型来讲解表格，那么本文也显得太短了。用户可以看到在创建表格的时候创建数据不是很方便的。除此之外创建二维数据也是一件不简单的编程过程。但是 Table 的 JVC 模型妥善地解决了这个问题。通过单独地定义该模型，使得模型的生命周期不影响表格的代码。

通常，用户可以通过扩展 AbstractTableModel 类来创建表格模型。AbstractTableModel 有几个方法需要实现。

- public int getRowCount( );
- public int getColumnCount( );
- public Object getValueAT(int row, int column);

用户应该能够确切地猜出如何实现这些方法。另一个表格模型 DefaultTableModel 扩展了 AbstractTabel 并增加了编辑表格的能力。程序清单 24.6 使用扩展了 DefaultTableModel 类的类修改了前面的例子。注意 JTabel 使用新类(TableModel)代替了固定的字符串。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.table.*;
import com.sun.java.swing.event.*;

public class TableExample2 extends JPanel{
  static JFrame myFrame;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
  public TableExample2(){
    setLayout(new BorderLayout());
    JTable table = new JTable(new TableModel());
    JScrollPane pane = JTable.createScrollPaneForTable(table);
    add(pane);
  }
  public static void main(String args[]){
    myFrame = new JFrame("Table Example #2");
    TableExample2 tableExample = new TableExample2();
    myFrame.getContentPane().add("Center", tableExample);
    myFrame.setSize(400,250);
    myFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {System.exit(0);}
      });
    myFrame.setVisible(true);
  }
}

class TableModel extends DefaultTableModel{
  String data[][] = {{"1","Introduction"},{"2","What Java can Do for You"},{"3","JAVA Design"}
➡,{"4","Installing JAVA"},{"5","JDK tools"},{"6","Object-Oriented Programming"},{"7","Hello world"}
➡,{"8","Data Types"},{"9","Methods"},{"10","Using Expressions"}};
  String columnNames[] = {"Chapter Number","Chapter Title"};

  public int getRowCount(){
    return data.length;
  }

  public int getColumnCount(){
    return data[0].length;
  }

  public Object getValueAt(int row,int column){
    return data[row][column];
  }

  public String getColumnName(int column){
    return columnNames[column];
  }

  public void setValueAt(Object value, int row, int column){
    if (value instanceof String){
      data[row][column] = (String)value;
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        //Make sure you fire the table changed data so the view etc. will get notified of the change
        fireTableChanged(new TableModelEvent(this,row,row,column));
    }
}
```

24.4.3　单元编辑器

当用户运行 TableExample2 时将会发现，如果用户试图在某个域中输入数据时将会改变文本域，并且用户能够以此来修改该值。这是非常有用的。但是用户是否希望为程序的使用者提供几个选项呢？比如使用组合框？好，表格支持单元编辑器的概念就是为了这个目的而设置的。单元编辑器提供有限的单元编辑控制。考虑到本章篇幅的限制，本文只讲述如何增加两栏：一个是组合框，另一个是复选框。

程序清单 24.7 显示了如何为单元设置一个新的组合框栏。另外它还演示了布尔变量也可以用复选框来表示。

程序清单 24.7　为 TableExample 增加一个组合框栏

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.table.*;
import com.sun.java.swing.event.*;

public class TableExample3 extends JPanel{
  static JFrame myFrame;


  public TableExample3(){
    setLayout(new BorderLayout());
    JTable table = new JTable(new TableModel());
    JScrollPane pane = JTable.createScrollPaneForTable(table);
    add(pane);

    JComboBox comboBox = new JComboBox(new String[]{"Low","Medium","High"});
    TableColumn priorityColumn = table.getColumn("Priority");
    priorityColumn.setCellEditor(new DefaultCellEditor(comboBox));

  }
  public static void main(String args[]){
    myFrame = new JFrame("Table Example #3");
    TableExample3 tableExample = new TableExample3();
    myFrame.getContentPane().add("Center",tableExample);
    myFrame.setSize(400,250);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        myFrame.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e) {System.exit(0);}
        });
        myFrame.setVisible(true);
    }
}

class TableModel extends AbstractTableModel{
    String data[][] = {{"1","Introduction","High"},
                {"2","What Java can Do for You","Low"},
                {"3","JAVA Design","Low"},{"4","Installing JAVA","Low"},
                {"5","JDK tools","Low"},
                {"6","Object-Oriented Programming","Low"},
                {"7","Hello world","Low"},{"8","Data Types","Low"},
                {"9","Methods","Low"},{"10","Using Expressions","Low"}};
    Boolean doneFlags[] = {new Boolean(false),new Boolean(false),
                new Boolean(false),new Boolean(false),new Boolean(false),
                            new Boolean(false),new Boolean(false),
                new Boolean(false),new Boolean(false),new Boolean(false)};
    String columnNames[] = {"Chapter Number","Chapter Title","Priority","Done?"};

    public int getRowCount(){
        return data.length;
    }

    public int getColumnCount(){
        return columnNames.length;
    }

    public Object getValueAt(int row,int column){
        if (column < 3)
            return data[row][column];
        else
            return doneFlags[row];
    }

    public String getColumnName(int column){
        return columnNames[column];
    }

    public void setValueAt(Object value, int row, int column){
        if (value instanceof String){
            data[row][column] = (String)value;
        }else if (value instanceof Boolean){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        doneFlags[row] = (Boolean) value;
      }
      fireTableChanged(new TableModelEvent(this,row,row,column));
    }
    public boolean isCellEditable(int row, int col) {return true;}
    public Class getColumnClass(int c) {return getValueAt(0, c).getClass();}
  }
```

24.5　树

使用图形用户界面设计的树状显示已经有多年了，树允许用户很容易地显示 n 维数据。树状显示文件列表非常流行。除此之外树状显示在其它情况下也很有用。JFC 新增加的树类叫做 JTree，它具有与 JTable 和 JList 相同的数据控制工具。

与表格不同，树不是从简单的字符串数组开始创建的。如果用户想这样做，那么最终将会做成一个非常漂亮的列表框。如果用户希望预定义根，那么这种方法是非常适合的，但因为这并不常用，所以此处不作这方面的介绍。

24.5.1　树结点

树是由结点组成的。每个节点既可以代表叶子，也可以代表枝干。叶是一种特殊的元素，它没有子元素或孩子。换句话说，叶子不能“打开”和“关闭”，如果用树的术语讲，就是“展开”和“衰萎”。枝干正好相反，它可以有孩子并且可以展开和衰萎显示这些孩子。孩子既可以是枝干也可以是叶子。实际上枝干和叶子都是节点，除了叶子没有孩子之外，叶子和枝干没有什么差异。

通常，用户将会发现 DefaultMutableTreeNode 类可以提供用户创建树节点的所有工具。程序清单 24.8 展示了如何使用 DefaultMutableTreeNode 建造树。图 24.10 显示了这段代码运行的结果。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.tree.DefaultMutableTreeNode;

public class TreeExample extends JPanel{
  static JFrame myFrame;

  public TreeExample(){
    setLayout(new BorderLayout());
    DefaultMutableTreeNode rootNode = createNodes();
    JTree tree = new JTree(rootNode);
    tree.setRootVisible(true);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    JScrollPane pane = new JScrollPane();
    pane.setViewportView(tree);
    add(pane);
  }

  public DefaultMutableTreeNode createNodes(){
    DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode ("Java Stuff");
    DefaultMutableTreeNode resources = new DefaultMutableTreeNode ("Resources");
    DefaultMutableTreeNode tools = new DefaultMutableTreeNode ("Tools");
    rootNode.add(resources);
    rootNode.add(tools);
    DefaultMutableTreeNode webSites = new DefaultMutableTreeNode ("Web Sites");
    DefaultMutableTreeNode books = new DefaultMutableTreeNode ("Books");
    resources.add(webSites);
    resources.add(books);

    DefaultMutableTreeNode magazines = new DefaultMutableTreeNode ("Magazines");
    webSites.add(new DefaultMutableTreeNode ("JavaSoft"));
    webSites.add(new DefaultMutableTreeNode ("Gamelan"));
    webSites.add(magazines);
    magazines.add(new DefaultMutableTreeNode ("Javology"));
    magazines.add(new DefaultMutableTreeNode ("JavaWorld"));

    books.add( new DefaultMutableTreeNode ("Special Edition Using Java 1.2"));

    tools.add(new DefaultMutableTreeNode ("JBuilder"));
    tools.add(new DefaultMutableTreeNode ("Visual J++"));
    tools.add(new DefaultMutableTreeNode ("Visual Age for Java"));
    tools.add(new DefaultMutableTreeNode ("Apptivity"));
    return rootNode;
  }

  public static void main(String args[]){
    myFrame = new JFrame("Tree Example");
    TreeExample treeExample = new TreeExample();
    myFrame.getContentPane().add("Center",treeExample);
    myFrame.setSize(400,250);
    myFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {System.exit(0);}
      });
    myFrame.setVisible(true);
  }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

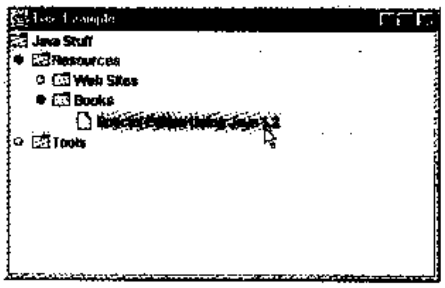


图 24.10　用户可以随意地展开和关闭树

理解 TreeExample　首先让我们看一下构造器。该树是使用 rootNode 对象创建的。后面将介绍 rootNode 是如何创建的，但重要的是要理解用户只传递了树的根节点。

用户看到的下一行代码将根结点设置为可视的。这实际上是缺省值，这里这样做是为了说明根节点通常要可视。用户可以看到根节点没有任何兄弟，这也意味着用户可以创建不提供任何信息的根节点。所以用户可以使用 tree.setRootVisible(false)把根设成不可视的。

构造器最后一个有意思的部分是像 JList 和 JTable 一样，JTree 也不自己处理滚动。和 JList 和 JTable 一样，JTable 实现了 Scrollable 接口。所以用户可以创建一个滚动窗格并把它加在上面。

创建节点　createNodes( )方法负责实际地创建树结点。用户实际要做的就是把每个分枝分好层。用户首先创建根结点，然后创建根节点的孩子。然后为根节点的孩子创建结点并把这些节点加到父节点上，依此类推。

24.5.2　树模型

当用户创建树时知道节点信息可以使节点不多的树工作得很好，如程序清单 24.8 所示。但是当用户在创建完树之后需要插入和删除节点时程序清单 24.8 中的程序就显得不够用了。原因是虽然用户仍然可以在可视的节点上调用 add( )方法，但是树并不知道该信息，所以树的可视部分没有被刷新。

像表格的模型一样，树模型也可以随时操作和存储各种值。关于处理 TreeModel 的细节超出本书的介绍范围，所以我们只介绍如何增加和删除节点。

程序清单 24.9 演示了该模型的使用。在窗口的顶部有一个文本域，它允许用户使用文本域中的值在当前选中的项目上增加新结点，在窗口的底部有一个按钮。它可以把选中的节点从树中删除掉(见图 24.11)。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.tree.DefaultMutableTreeNode;
import com.sun.java.swing.tree.DefaultTreeModel;
import com.sun.java.swing.tree.TreePath;

public class TreeExample extends JPanel{
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    static JFrame myFrame;
    JTextField tf;
    JTree tree;

    public TreeExample(){
      setLayout(new BorderLayout());
      tf= new JTextField();
      tf.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent ae){
        addTextFieldValue();
        }
        });
      add(tf,"North");

      DefaultMutableTreeNode rootNode = createNodes();
      tree = new JTree(rootNode);
      tree.setRootVisible(true);
      JScrollPane pane = new JScrollPane();
      pane.setViewportView(tree);
      add(pane,"Center");

      JButton remove = new JButton("Remove");
      remove.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent ae2){
        removeSelectedNode();
        }
        });
      add(remove,"South");
    }

    public void removeSelectedNode(){
      TreePath selectionPath = tree.getSelectionPath();
      DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode)
➡SelectionPath.getLastPathComponent();
      ((DefaultTreeModel)tree.getModel()).removeNodeFromParent(selectedNode);
    }
    public void addTextFieldValue(){
      DefaultMutableTreeNode newNode = new DefaultMutableTreeNode(tf.getText());
      TreePath selectionPath = tree.getSelectionPath();
      DefaultMutableTreeNode selectedNode = (DefaultMutableTreeNode)
➡selectionPath.getLastPathComponent();
      ((DefaultTreeModel)tree.getModel()).insertNodeInto(newNode,
➡ selectedNode, selectedNode.getChildCount());
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public DefaultMutableTreeNode createNodes(){
    DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("Java Stuff");
    DefaultMutableTreeNode resources = new DefaultMutableTreeNode("Resources");
    DefaultMutableTreeNode tools = new DefaultMutableTreeNode("Tools");
    rootNode.add(resources);
    rootNode.add(tools);
    DefaultMutableTreeNode webSites = new DefaultMutableTreeNode("Web Sites");
    DefaultMutableTreeNode books = new DefaultMutableTreeNode("Books");
    resources.add(webSites);
    resources.add(books);

    DefaultMutableTreeNode magazines = new DefaultMutableTreeNode("Magazines");
    webSites.add(new DefaultMutableTreeNode ("JavaSoft"));
    webSites.add(new DefaultMutableTreeNode ("Gamelan"));
    webSites.add(magazines);
    magazines.add(new DefaultMutableTreeNode ("Javology"));
    magazines.add(new DefaultMutableTreeNode ("JavaWorld"));

    books.add(new DefaultMutableTreeNode("Special Edition Using Java 1.2"));

    tools.add(new DefaultMutableTreeNode ("JBuilder"));
    tools.add(new DefaultMutableTreeNode ("Visual J++"));
    tools.add(new DefaultMutableTreeNode ("Visual Age for Java"));
    tools.add(new DefaultMutableTreeNode ("Apptivity"));

    return rootNode;
}

public static void main(String args[]){
    myFrame = new JFrame("Tree Example");
    TreeExample treeExample = new TreeExample();
    myFrame.getContentPane().add("Center",treeExample);
    myFrame.setSize(400,250);
    myFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {System.exit(0);}
      });
    myFrame.setVisible(true);
}
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

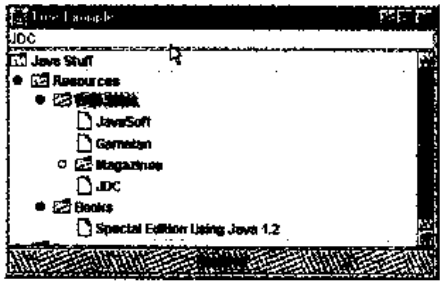


图 24.11 用户可以使用 TextField 文本域在树中增加节点，可以使用 remove 按钮从树中删除节点

24.5.3 理解新的 TreeExample

本例的大部分留给用户自己浏览和领会，只有两个方法需要在这里探究，它们是 removeSelctedNode( )和 addText FieldValue( )。

24.5.4 removeSelctedNode

removeSelctedNode 被设计用来删除树中被选中的节点。为了删除节点，必须首先获得被选中的节点。该树可以通过 getSelectionPath( ) 方法为用户提供该节点的路径。该方法不返回节点本身，而是返回一个名叫 TreePath 的新类。TreePath 包括从根节点到被选中节点的所有节点。这样做对后续编程是很方便的。因为用户可以存储该路径并确保以后能够使用树的 expandPath( )方法将所有的这些文件夹打开以显示该节点。或者按别的方式使用它。

在本例中，树路径用于获得被选中的节点，不要求该路径的最后组件。GetLastPathComponent( )方法实际返回一个 Object，所以用户可以注意到该结果被强制为 DefaultMutableTreeNode 类型。

用户知道了节点后，就可以删除它了。正如前面提到那样，用户希望删除树中该节点的部分是模型。该模型将通知所有其它必要的组件。幸运的是，DefaultMutableTreeNode 正好拥有一个用于删除该节点的方法，该方法的名字为 removeNodeFromParent，它就可以做这项工作。

> **注意**：在本例中，因为用户使用的不是特殊模型，而只是缺省模型，所以用户可以将 getModel( )强制成为 DefaultTreeModel。如果用户将它改变成自定义模型，则需要调整该代码。
>
> 用户也可能希望知道何时从树中删除该节点，节点本身并不实际“执行”删除动作。如果用户希望把它放在别处，或者放在别的树上，那么用户需要采取安全措施来调整它。

24.5.5 AddTextFieldValue( )

在该类中另一个令用户感兴趣的方法是 addTextFieldValue( )方法。该方法使用文本域中的文本创建一个新节点并把新节点作为孩子增加到被选中节点上。

增加节点的第一个任务是获得它的父节点。在前面的部分用户已经看到了执行该任务的大部分代码是如何工作的。该方法的最后一行有些特别。DefaultTreeModel 的 insertNoedInto( )方法使用了三个参数。第一个参数是新节点。第二参数是新节点的父节点，最后一个参数

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

是该节点加在列表中的位置。在本例中，用户可以将它作为最后一个孩子加到该节点上，但是用户也可以将它作为第一个元素加进节点或者是加在用户希望的任何位置上。用户也可以按照自己的意图加进去看一看效果。

24.6　使用 JEditor 显示 HTML 文件

另一个加在 JFC 中的重大创新之一是一个编辑器。JEditor 允许用户显示格式化文本，该文本带有使文本拥有特殊属性的标记，比如居中和斜体。HTML 和 RTF 文件编辑器工具是标准的 JFC 例子。用户最好是自己定义 EdetorKit 类的实现并指定用户自己的文件格式。

在程序清单 24.10 中，用户可以看到 JEditor 被用来创建用户可以调用的基本的网页浏览器。图 24.12 显示了该代码的运行结果。

[illegible] HTMLView.java ——使用 JEditor 的浏览器

```
import com.sun.java.swing.*;
import com.sun.java.swing.event.*;
import java.awt.BorderLayout;
import java.awt.event.*;
import java.io.IOException;
import java.util.Date;
import java.net.URL;
import java.net.MalformedURLException;

public class HTMLView extends JPanel{
  // The initial width and height of the frame
  public static int WIDTH = 600;
  public static int HEIGHT = 400;

  protected JEditorPane _view;
  protected JTextField _commandLine;

  public HTMLView(){
    _view = new JEditorPane();
    _view.setEditable(false);
    _view.addHyperlinkListener(new HyperlinkListener(){
      //This is called if a hyperlink is clicked on
      public void hyperlinkUpdate(HyperlinkEvent event){
      setURL(event.getURL());
      }
      });

    setLayout(new BorderLayout());
    add(_view,"enter");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    //Add the location line
    _commandLine = new JTextField();
    add(_commandLine,"North");
    _commandLine.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
        try{
          URL newURL = new URL(ae.getActionCommand());
          setURL(newURL);
        }catch (MalformedURLException mue){
          System.out.println("The URL you clicked on appears incorrect:" + mue);
        }
      }
    });
  }

  public void setURL(URL newURL){
    try{
      _view.setPage(newURL);
      _commandLine.setText(newURL.toExternalForm());
    }catch (IOException ioe){
      System.out.println("Error :" + ioe);
    }
  }

  public static void main(String s[]) {
    HTMLView panel = new HTMLView();

    JFrame frame = new JFrame("HTML Example");
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {System.exit(0);}
      });
    frame.getContentPane().add("Center", panel);
    frame.pack();
    frame.setVisible(true);
    frame.setSize(WIDTH,HEIGHT);
  }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

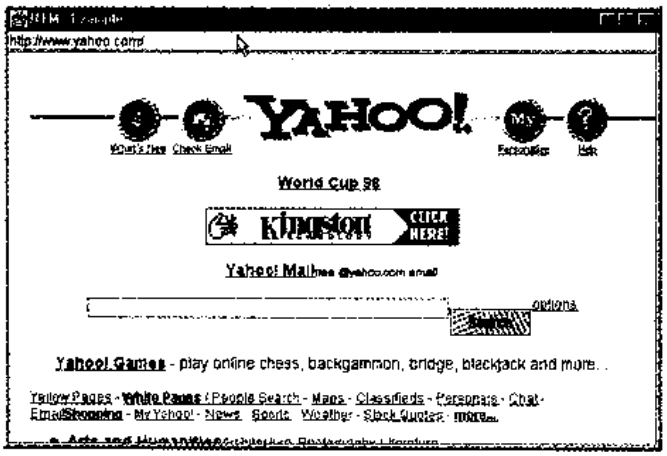


图 24.12　HTMLView 可以显示像 Yahoo.com 这样的网页

24.6.1　理解 HTMLView

HTMLView 有两个不同的组件。JEditor 和 JTextField。编辑器显示当前的网页并允许用户单击任何超链。文本域允许用户指定网页，就像 Netcape Navigator 或 Internet Explorer 的地址栏一样。

24.6.2　监听超链事件

当用户在 JEditor 中单击超链时，JEditor 将产生一个名叫 HyperlinkEvent 的事件。一般地讲，当浏览器中发生这个事件时，用户希望将编辑器当前页改变为由该超链指向的 URL。在用户使用自己的小浏览器获得该结果之前，首先需要捕捉该事件。和使用所有事件一样，用户可以使用监听器捕捉该事件，而该监听器需要实现 HyperlinkListener 接口。正如用户在程序清单 24.10 的下述代码片段中看到的那样，HyperlinkListener 接口只有一个方法 hyperlinkUpdate()。

```
_view.addHyperlinkListener(new HyperlinkListener( ){
        //如果超链被单击就调用该方法
        public void hyperlinkUpdate(HyperlinkEvent event){
        setURL(event.getURL( ) );
        }
});
```

24.6.3　设置新页

要想为 JEditor 设置新页，用户需要使用 setPage( )方法。做这件事情的技巧是 setPage( )方法可以抛出一个 IOException 异常，用户可以捕捉该 IOException 异常，如下面的代码片段所示。该代码片段是从程序清单 24.10 中摘录出来的。

```
public void setURL(URL newURL){
    Try{
            _view.setPage(newURL);
            _commandLine.setText(newURL.toExternalForm( ));
    }catch (IOException ioe){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            sysetm.out.println("Error:" + ioe);
        }
    }
```

24.6.4 JTextField

JFC 的 JTextField—JTtextField 的工作方式与 AWT 的 JTextField 工作方式非常类似。正如在程序清单 24.10 里看到的那样。当用户按下 Enter 键时 ActionListeners 可以接收到通知，用户可以使用该通知调用与我们在 HyperlinkListener 接口中使用 setURL 方法相同的方式来改变网页。

24.7 创建图标

在本章和第 23 章中我们都使用了图标。但是到目前为止我们还没有使用过从图形文件中创建的图标。正像用户可以猜测到的那样，图形图标并不总像图像文件那样高效。这主要是从尺寸和执行效率的观点上讲的，另外，调整成用户需要的尺寸和颜色也是比较困难的。虽然使用图像允许尺寸和颜色的变化，但是结果不可能不产生图像质量损失，并且这些方法的运行速度绝不可能太快。

优先选择的创建简单图标的机制是实现 Icon 接口。Icon 接口有三个方法：

```
public abstract void paintIcon(Component c, Graphics g, int x, int y)
```

该方法在 Graphics 对象上的 x，y 坐标处绘制图标。

```
public abstract int getIconWidth( )
```

该方法返回图标的宽度。注意这个值是固定的，并且以后也不能修改它。

```
public abstract int getIconHeight( )
```

该方法返回图标的高度。注意这个值是固定的，并且以后也不能修改它。

前两章用户已经熟悉了 ImageIcon，当用户调用 paintIcon 方法时，ImageIcon 就简单地绘制该图像。在多数情况下，它花费很少的时间，占用很少的文件空间就可以绘制出图标。下面的程序清单 24.11 显示了一个绘制公牛眼睛图标的示例。图 24.13 显示了该代码的结果。

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
import com.sun.java.swing.border.*;

public class IconButton extends JPanel {
  static JFrame myFrame;
  protected JLabel label;
  JPopupMenu pm;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
  public IconButton(){
    JButton hello = new JButton("Hello");
    hello.setMnemonic('h');
    hello.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
      label.setText("Hello World!");
      }
      });

    //set the background of the button
    hello.setBackground (SystemColor.control);

    //Set the icon for the button
    Icon icon = new MyFirstIcon ();
    hello.setIcon(icon);

    add(hello);

  }

  public static void main(String args[]){
    myFrame = new JFrame("Hello World!");
    IconButton ib = new IconButton();
    myFrame.getContentPane().add("Center",ib);
    myFrame.setSize(300,75);
    myFrame.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent e){System.exit(0);}
      });
    myFrame.setVisible(true);
  }
}

class MyFirstIcon implements Icon{

  static private final Dimension size = new Dimension( 20, 20 );

  public void paintIcon(Component c, Graphics g, int x, int y) {
    //translate g to the x,y coordiantes so you don't have to deal
    //with it through out the rest of the method
    g.translate( x, y );

    int right = size.width;
    int bottom = size.height;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        int ringWidth = right/10;
        int ringHeight = bottom/10;


        // Draw white fill
        g.setColor( Color.red );
        g.fillOval(0,0,right,bottom  );
        // Draw outer ring
        g.setColor( Color.white );
        g.fillOval(ringWidth,ringHeight,right - ringWidth * 2,bottom-ringHeight * 2);

        // Draw middle white ring
        g.setColor( Color.red);
        g.fillOval(ringWidth * 2,ringHeight * 2,right-ringWidth * 4,bottom-ringHeight * 4);

        //draw middle ring
        g.setColor( Color.white );
        g.fillOval(ringWidth * 3,ringWidth * 3,right-ringWidth * 6,bottom-ringHeight * 6);

        // Draw middle white ring
        g.setColor( Color.red);
        g.fillOval(ringWidth * 4,ringHeight * 4,right - ringWidth * 8,bottom-ringHeight * 8);

        //draw inner ring
        g.setColor( Color.white );
        g.fillOval(ringWidth * 5,ringWidth * 5,right - ringWidth * 10,bottom-ringHeight * 10);


        //Translate g back to where it was when we started the
        //method
        g.translate( -x, -y );
    }


    public int getIconWidth() { return size.width; }
    public int getIconHeight() { return size.height; }

}
```



图 24.13 按钮上拥有一个独特的图标

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

24.8　JFC 小应用程序

既然用户对所有这些 JFC-Swing 要素有了浓厚的兴趣，那么用户现在就可以使用 JFC 来编写一些小应用程序了。在用户开始编写小应用程序之前，用户需要考虑一个重要的技术问题：JFC 是一个很大的包；用户能使用得了它吗？如果用户正在编写一个企业网小应用程序，这可能不会成为问题，但是如果用户正在编写国际互联网上的应用程序，那么用户就需要告诉程序的使用者必须在每次运行该应用程序时下载这个巨大的文件了。

为了修改用户的小应用程序，只需要做两件事情。首先，扩展 JApplet，而不是 Applet，其次是将组件增加到内容窗格上，而不是直接加在小应用程序上。

程序清单 24.12 显示了如何使用按钮建造 HelloWorld 小应用程序。注意 feet.gif 图标也可以用在这里。

```
import java.awt.*;
import java.awt.event.*;
//import com.sun.java.swing.*;
import com.sun.java.swing.*;

public class HelloWorldJFCApplet extends JApplet {
  static JFrame myFrame;

  public void init(){

    Icon icon = new ImageIcon("feet.gif");
    JLabel label = new JLabel("Hello World!",icon,SwingConstants.RIGHT);

    //For an Applet you must add everything to the content pane.
    Container panel = getContentPane();
    panel.setLayout(new FlowLayout());
    panel.add(label);
  }
}
```

配置 Netscape 和 Internet Explorer 以使用 JFC　在 Navigator 或 Internet Explorer 下使用 JFC 最好的办法是使用 Java 插件。插件包括 Swing API 和确保使用真正的 Java 虚拟机。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第25章 图 像

本章内容：

25.1 在屏幕上绘制图像
25.2 生产者、消费者和观察者
25.3 图像过滤器
25.4 从内存拷贝图像
25.5 将图像拷贝到内存
25.6 颜色模式

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

25.1　在屏幕上绘制图像

Java 操作图像的方法与许多常规的图形系统有所不同。为了支持基于网络的操作，Java 必须支持渐进加载图像的绘图法。用户不希望小应用程序坐等着所有图像下载完毕。Java 的生产者—消费者模型采用了渐进的图像加载。Java 使用了过滤器的概念使用户能够在从生产者传送给消费者的过程中修改图像。这种处理图像的方式看起来有些陌生，但它是很强大的。

仅仅在屏幕上绘制基本图像在 Java 中是很简单的。Graphics 类提供了一个比较方便的方法叫做 drawImage( )，它就是用于绘制基本图像的。在前面章节的几个程序中都使用了 drawImage 方法。在第 15 章“高级小应用程序代码”的“在小应用程序中增加图像”部分中用户了解到了从 URL 中加载图像和使用 paint( )方法绘制它们。paint( )方法可以做比 drawImage 更多的工作，它允许用户动态地缩放图像。从 Java1.1 开始，用户就能够使用 drawImage 方法修剪和旋转图像了。

程序清单 25.1 显示了如何使用 drawImage 方法放大一倍图像尺寸后显示在屏幕上。如图 25.1 所示。

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Image;
import java.net.URL;
import java.net.MalformedURLException;

public class ScaleImage extends Applet{
  Image img;

  public void init(){
    try{
      img = getImage (new URL(getDocumentBase(),"MagnaHeader.gif"));
    }catch (MalformedURLException e){
      System.out.println("URL not valid:"+e);
    }
  }

  public void paint (Graphics g){
    g.drawImage (img,0,0,img.getWidth(null) * 2,img.getHeight(null) * 2,this);
  }
}
```

从 Java1.1 开始增加了另一个 drawImage 方法，该 drawImage 方法提供了更多的功能。它有下列特征：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

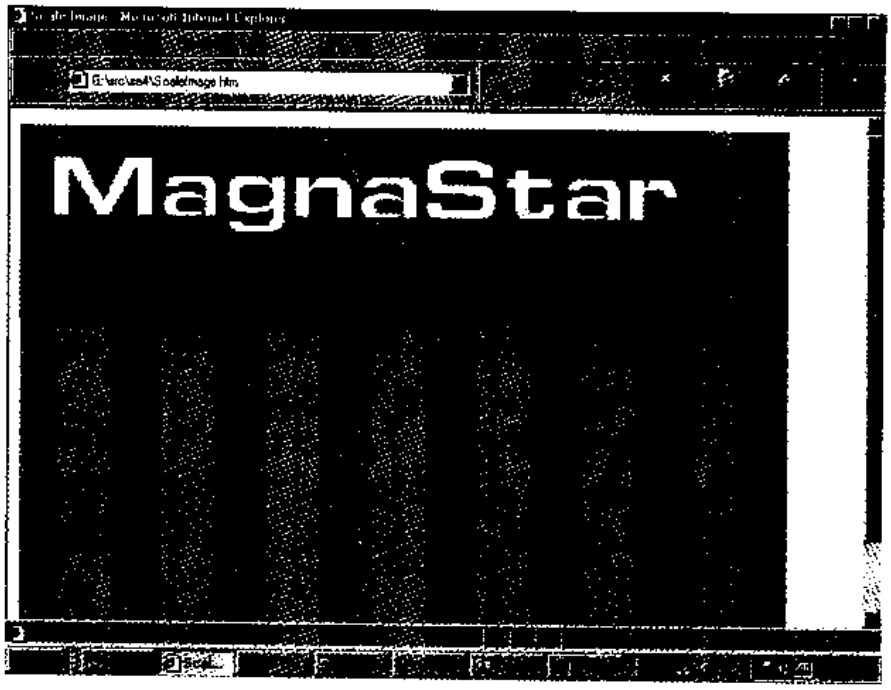


图 25.1 用户可以在屏幕上绘制简单的图像并加倍放大它的尺寸

```
public abstract boolean drawImage(Image img,
        int dx1,int dy1, int dx2, int dy2,
        int sx1, int sy1, int sx2, int sy2,
        ImageObserver observer)
```

这个 drawImage 方法工作起来比它的兄弟稍微困难一点。首先第一个参数集(dx 变量)指示的不是开始的 x 坐标和 y 坐标以及高度和宽度,而是图像左上角的 x,y 坐标和图像右下角的 x,y 坐标。这意味着用户可以直接执行实际的轴转换。第二个参数集(sx 参数)指示在原坐标体系中的起始和结束的坐标,这套参数允许用户任意地截选图像。

现在让我们看一下实际上如何使用 drawImage 方法。程序清单 25.2 显示了如何使用方法翻转一幅图像,如图 25.2 所示。

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Image;
import java.net.URL;
import java.net.MalformedURLException;

public class FlipImage extends Applet{
  Image img;

  public void init(){
    try{
      img = getImage (new URL(getDocumentBase(),"MagnaHeader.gif"));
    }catch (MalformedURLException e){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            System.out.println("URL not valid:" + e);
        }
    }

    public void paint (Graphics g){
        g.drawImage(img,0,
                    img.getHeight(null), img.getWidth(null), 0,0,0,
                    img.getWidth(null),img.getHeight(null),this);
    }
}
```





图 25.2　drawImage 翻转图像

下面让我们看一下如何使用 sx 变量从该图像的中心截选一部分。程序清单 25.3 截取了四分之一的图像并翻转显示在屏幕上(见图 25.3)。

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Image;
import java.net.URL;
import java.net.MalformedURLException;

public class FlipCropImage extends Applet{
    Image img;

    public void init(){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    try{
      img = getImage(new URL(getDocumentBase(),"MagnaHeader.gif"));
    }catch (MalformedURLException e){
      System.out.println("URL not valid:" + e);
    }
  }

  public void paint (Graphics g){
    g.drawImage (img,0, img.getHeight(null)/2,img.getWidth(null)/2,
                 ➡0,img.getWidth(null)/4, img.getHeight(null)/4,
                 ➡img.getWidth(null) * 3/4 ,img.getHeight(null) * 3/4, this);
  }
}
}
```

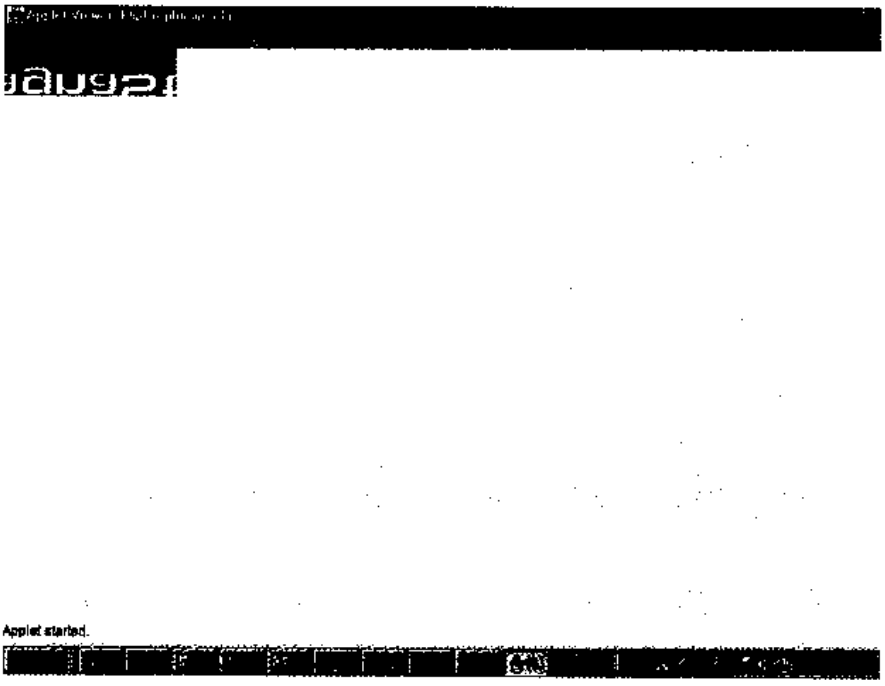


图 25.3 drawImage 允许用户截取图像的一部分

25.2 生产者、消费者和观察者

Java 操作图像的模型比其它的模型复杂。在该模型中 Java 使用了图像生产者和图像消费者的概念。举例来说图像生产者可以是负责从网上拿取图像的对象,也可以是简单的代表图像的字节数组。图像生产者可以认为是图像数据的源。图像消费者是使用图像数据的对象。

典型的图像消费者是低级的绘画例程,它负责在屏幕上显示图像。关于生产者 - 消费者模型较有趣的东西是生产者处于控制地位。ImageProducecr 使用 ImageConsumer 的 setPixels

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

方法向消费者描述图像。

这个机制的最好方式是跟踪一个从网络上加载图像的过程。首先 ImageProducer 开始读取图像。从图像中读取的第一个东西是图像的宽度和高度。生产者使用 SetDimension 方法通知它的消费者(注意:一个生产者可以为多个消费者提供服务)图像的尺寸。图 25.4 阐述了 ImageProducer 和 ImageConsumer 之间的关系。

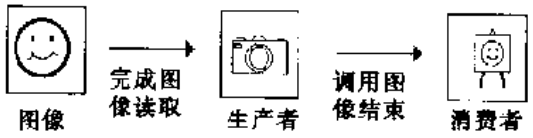


图 25.4　ImegeProducer 从图像文件中读取图像的尺寸并将该信息传递给 ImegeProducer

接下来生产者读取图像的颜色映射。从颜色映射中生产者确定图像使用的颜色模式,然后调用各消费者的 setColorModel 方法。图 25.5 阐述了生产者如何向消费者传递颜色信息。

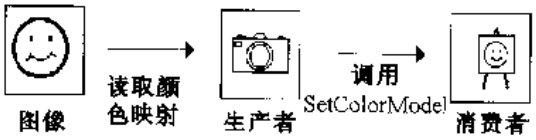


图 25.5　生产者使用 setColorModel 方法向消费者传送颜色信息

生产者调用每个消费者的 setHints 方法告诉消费者它打算如何传递图像的像素。这使得消费者可以尽可能地优化它的像素处理。某些提示值是:

ImageConsumer.RANDOMPIXELORDER.

ImageConsumer.TOPDOWNLEFTRIGHT.

ImageConsumer.COMPLETESCANLINES.

ImageConsumer.SINGLEPASS 和 ImegeConsumer.SINGLEFRAM。图 25.6 阐述了生产者如何向消费者传递提示。



图 25.6　生产者将提示传递给消费者以说明它将如何发送像素

现在生产者终于开始"生产"像素了,它调用消费者的 setPixels 方法传递该图像。这个工作是在许多次调用中完成的,特别是当对大图像每次只传递一个描述行时。它也可以是单个调用,但必须是消费者将图像作为单传递(ImegeConsumer.SINGLEPASS)来传送。图 25.7 显示了生产者向消费者传递像素信息的过程。

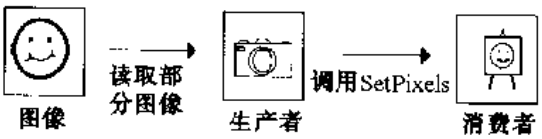


图 25.7　生产者使用 setPixels 方法传递像素信息给消费者

最后,生产者调用消费者的 imageComplete 方法说明图像已传送完毕。如果传递失败,例如正在传递数据时网络联系中断了,那么将调用带有参数 ImageCosumer.IMAGEERROR 或 ImageCosumer.IMAGEABORT 的 imageComplete 方法。另一种可能的情况是该图像是一个

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

多帧图像(一种动画形式)中的一部分并且还有其它的帧要传来。这时应使用 ImageCosumer.SINGLEFRAMEDONE 参数做出标志。当一切事情都已真正完成时,将调用带有参数 ImageCosumer.STATICIMAGEDONE 的 imageComplete 方法。图 25.8 显示了生产者正全神贯注地将图像传递给消费者。



图 25.8 生产者使用 imagineComplete 方法告诉消费者它传递完了图像

这个方法允许 Java 能够有效地加载图像;它在开始别的工作之前不必停下来等待图像的传递。ImegeObserver 接口作为一种"有趣的第三者"联系着生产者 - 消费者接口。它可以使某个对象当生产者发布出某些关于该图像的新信息时受理刷新。

用户可以回想起来:当用户使用 drawImage 方法时,可以将 this 作为最后一个参数传递。用户实际上可以给 drawImage 方法一个对 ImageObserver 的引用。Applet 类实现了 ImageObserver 接口。ImageObserver 接口包含了一个唯一的方法名叫 imageUpdate:

```
boolean imageUpdate(Image img, int flags, int x, int y,int width, int height)
```

并非所有传递给 imageUpdate 方法的信息总是有效的。flags 参数是告诉关于图像的哪些信息有效的概括标志。下面列出的是可能的标志:

ImageObserver.WIDTH	宽度值目前是有效的
ImageObserver.HEIGHT	高度值目前是有效的
ImageObserver.PROPERTIES	图像属性目前是可用的
ImageObserver.SOMEBITS	更多的像素是可用的
	(x,y,宽度和高度指示由目前的像素捆绑的矩形)
ImageObserver.FRAMEBITS	另外一个目前可用的完整框架
ImageObserver.ALLBITS	被完全加载的图像
ImageObserver.ERROR	有一个图像加载错误
ImageObserver.ABORT	图像加载被中止

这些标志是被加在一起的,所以下面的代码使用 imageUpdate 方法测试 WIDTH 标志:

```
if((flag&Imageobsever.WIDTH)! =0)//目前宽度是有效的|
```

25.3 图像过滤器

Java 图像模型允许用户轻松地过滤图像。过滤器的概念类似于摄影中滤色镜的概念。它是位于图像消费者(胶片)和图像生产者(外部景象)之间的东西。过滤器在图像被传递给消费者之前修改图像。CropImageFilter 是一个预定义的过滤器,它把图像剪裁成一定尺寸(它只是整个图像的一部分)。用户可以通过把剪裁矩形的 x,y 坐标以及宽度和高度传递给构造器来创建 CropImageFilter 对象:

```
public CropImageFilter(int x,int y, int width, int height)
```

创建完图像过滤器之后,用户可以通过创建 FilteredImageSource 对象来将过滤器放在已

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

有图像源的顶上：

```
public FilteredImageSource(ImageProducer imageSource, ImageFilter filter)
```

程序清单 25.4 获得了一幅图像并对它使用 CropImageFilter 过滤器，结果只显示该图像的一部分。图 25.9 包含了这个小应用程序的输出，它显示了一个完整的图像和该图像的裁剪过的版本。

```
import java.awt.*;
import java.awt.image.*;
import java.applet.*;

// Example 25.4 - CropImage Applet
//
// This applet creates a CropImageFilter to create a
// cropped version of an image. It displays both the original
// and the cropped images.

public class CropImage extends Applet
{
  private Image originalImage;
  private Image croppedImage;
  private ImageFilter cropFilter;

  public void init()
  {
//Get the original image
      originalImage = getImage(getDocumentBase(), "samantha.gif");

//Create a filter to crop the image in a box starting at (25, 30)
//that is 75 pixels wide and 75 pixels high.

      cropFilter = new CropImageFilter(25, 30, 75, 75);

//Create a new image that is a cropped version of the original

      croppedImage = createImage(new FilteredImageSource(
        originalImage.getSource(), cropFilter));
  }

  public void paint(Graphics g)
  {
// Display both images
      g.drawImage(originalImage, 0, 0, this);
      g.drawImage(croppedImage, 0, 200, this);
  }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。



图 25.9 CropImage Filter 过滤器允许用户只显示图像的一部分

25.4 从内存拷贝图像

一种可能的图像生产者的类型是代表每个像素的颜色值的整数数组。MemoryImageSouree 类就是这种图像生产者。用户可以先创建内存图像，然后创建一个扮演该内存图像的图像生产者的 MemoryImageSource。然后再从 MemoryImageSource 创建一幅图像。MemoryImageSource 拥有许多构造器。使用所有这些构造器时，用户必须提供图像的宽度和高度，像素值数组、数组中第一个像素的启始偏移量和构成图像扫描选择位置数。像素值通常是每个像素的 RGB 值；但是，如果用户提供自己的颜色模式，那么像素值的含义就由该颜色模式来确定了。扫描行长度实际上与图像宽度是相同的。

但是有时像素值数组可能在扫描行末尾有额外的内容，这样扫描行的长度就会大于图像的宽度。用户不能使扫描行宽度小于图像宽度。用户也可以传递一个描述传递给图像消费者的图像属性的表。只有当图像消费者要求时才需要传递该属性。JDK 载有的消费者不要求任何属性。下面这些是 MemoryImageSource 的构造器：

public MemoryImageSource(int width,int height,ColorModel model,byte[] pixels,int startingOffset, int scanlineLength)

public MemoryImageSource(int width,int height,ColorModel model, byte[] pixels,int startingOffset,int scanlintLength,Hashtable properties)

public MemoryImageSource(int width,int height,ColorModel model,int[] pixels,int startingOffset,int scanlineLemgth)

public MemoryImageSource(int width,int height,ColorModel model,int[] pixels,int startingOffset,int scanlineLength,Hashtable properties)

public MemoryImageSource(int width,int height,int[] pixels,int startingOffset,int scanlineLength)

public MemoryImageSource(int width,int height,int[] pixels,int startingOffset,int scanlineLength.Hashtable properties)

程序清单 25.5 中的小应用程序创建了一个内存图像和一个 MemoryImageSource 并最终把该图像绘制在画图区内。图 25.10 显示了这个小应用程序的输出。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。



图 25.10　MemoryImageSource 类允许用户基于像素值创建自己的图像

```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;

// Example 25.5 - MemoryImage Applet
//
// This applet creates an image using an array of
// pixel values.

public class MemoryImage extends Applet
{
   private final static int b = Color.blue.getRGB();
   private final static int r = Color.red.getRGB();
   private final static int g = Color.green.getRGB();

// Create the array of pixel values.  The image will be 10x10
// And resembles a square bullseye with blue around the outside,
// green inside the blue, and red in the center.

   int pixels[] = {
      b, b, b, b, b, b, b, b, b, b,
      b, b, b, b, b, b, b, b, b, b,
      b, b, g, g, g, g, g, g, b, b,
      b, b, g, g, g, g, g, g, b, b,
      b, b, g, g, r, r, g, g, b, b,
      b, b, g, g, r, r, g, g, b, b,
      b, b, g, g, g, g, g, g, b, b,
      b, b, g, g, g, g, g, g, b, b,
      b, b, b, b, b, b, b, b, b, b,
      b, b, b, b, b, b, b, b, b, b};

   Image myImage;

   public void init()
   {
// Create the new image from the pixels array.  The 0, 10 means start
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// reading pixels from array location 0, and there is a new row of
// pixels every 10 locations.
        myImage = createImage(new MemoryImageSource(10, 10,
            pixels, 0, 10));
    }

    public void paint(Graphics g)
    {
// Draw the image.  Notice that the width and height we give for the
// image is 10 times its original size.  The drawImage method will
// scale the image automatically.
        g.drawImage(myImage, 0, 0, 100, 100, this);
    }
}
```

25.5 将图像拷贝到内存

PixelGrabber 类是与 MemoryImageSource 正好相反的类。它不是把整数数组转为图像，而是将图像转成整数数组。PixelGrabber 扮演一位 ImageConsumer。用户创建 PixelGrabber 时，可以给出用户希望的图像尺寸、用于存储图像像素的数组和从中获取像素的 ImageProducer。

为了抓取像素，用户必须通过传递用户要抓取的图像、用户要抓取的区域的 x 坐标和 y 坐标及高度和宽度、用来存放像素值的数组、像素值数组的偏移量和像素值数组的扫描行长度：

```
public PixelGrabber(Image image, int x, int y,int width,int heigh,int[] pixels,int startingOffset,int scanlineLength)
```

用户也可以提供一个代替图像的图像生产者：

```
public PixelGrabber(ImageProducer producer,int x,int y, int width,int height, int[] pixels, int startingOffset,int scanlineLength)
```

为了开始抓取像素，用户可以调用 grabPixels 方法。

```
public boolean grabPixels(long ms) throws InterruptedException
```

该方法开始抓取像素直到抓完所有像素为止。如果抓取像素成功，该方法返回 true。如果发生错误或中止运行，该方法返回 false。

用户可以使用 status 方法检查像素抓取的状态：

```
public synchronized int status()
```

由 status 返回的值包含有与 imageObserver 中的 imageUpdate 方法的 flags 参数相同的信息。基本上，如果 imageObserver.ABORT 位被设置为 1，那么像素抓取操作被中止；否则正常抓取。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果用户获取已有图像并修改它，那么 PixelGrabber 就显得很有用。程序清单 25.6 是一个小应用程序，它使用 PixelGrabber 将某个图像的像素抓取到一个数组中。然后它允许用户用一支颜色笔在用户想要着色的区域为图像的某个部分着色。为了重新显示该图像，小应用程序使用 MemoryImageSource 将像素数组还原成图像。该小应用程序在 486/100 机器上运行得相当慢，所以用户需要有足够的耐心。该小应用程序要求 Shape 类。

```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;

// Example 25.6 - Crayon Applet
//
// The Crayon applet uses the PixelGrabber to create an array of pixel
// values from an image.  It then allows you to paint the image using
// a set of crayons, and then redisplays the image using the
// MemoryImageSource.
// If you want to use other images with this applet, make sure that
// the lines are done in black, since it specifically looks for black
// as the boundary for an area.
// Also, beware, this applet runs very slowly on a 486/100

public class Crayon extends Applet
{
    private Image coloringBook;    // the original image
    private Image displayImage;    // the image to be displayed
    private int imageWidth, imageHeight;    // the dimensions of the image

// the following two arrays set up the shape of the crayons

    int crayonShapeX[] = { 0, 2, 10, 15, 23, 25, 25, 0 };
    int crayonShapeY[] = { 15, 15, 0, 0, 15, 15, 45, 45 };

// We use the ShapeObject class defined earlier so we can move the crayons
// to a new location easily.
    private ShapeObject crayons[];

// The color class doesn't provide a default value for brown, so we add one.
    private Color brown = new Color(130, 100, 0);

// crayonColors is an array of all the colors the crayons can be.  You can
// add new crayons just by adding to this array.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    private Color crayonColors[] = {
        Color.blue, Color.cyan, Color.darkGray,
        Color.gray, Color.green, Color.magenta,
        Color.orange, Color.pink, Color.red,
        Color.white, Color.yellow, brown };

    private Color currentDrawingColor;    // the color we are coloring with

    private int imagePixels[];    // the memory image of the picture

    boolean imageValid = false;    // did we read the image in o.k.?

// blackRGB is just used as a shortcut to get to the black pixel value
        private int blackRCB = Color.black.getRGB();

    public void init()
    {
        int i;
        MediaTracker tracker = new MediaTracker(this);

// Get the image we will color
        coloringBook = getImage(getDocumentBase(), "smileman.gif");

// tell the media tracker about the image
        tracker.addImage(coloringBook, 0);

// Wait for the image, if we get an error, flag the image as invalid
        try {
            tracker.waitForID(0);
            imageValid = true;
        } catch (Exception oops) {
            imageValid = false;
        }

// Get the image dimensions
        imageWidth = coloringBook.getWidth(this);
        imageHeight = coloringBook.getHeight(this);

// Copy the image to the array of pixels
        resetMemoryImage();

// Create a new display image from the array of pixels
        remakeDisplayImage();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// Create a set of crayons.  We determine how many crayons to create
// based on the size of the crayonColors array
            crayons = new ShapeObject[crayonColors.length];

            for (i=0; i < crayons.length; i++)
            {
// Create a new crayon shape for each color
                crayons[i] = new ShapeObject(crayonShapeX,crayonShapeY, crayonShapeX.length);

// The crayons are lined up in a row below the image
                crayons[i].moveShape(i * 30,imageHeight + 10);
        }
// Start coloring with the first crayon
        currentDrawingColor = crayonColors[0];
    }

// resetMemoryImage copies the coloringBook image into the
// imagePixels array.

        private void resetMemoryImage()
        {
            imagePixels = new int[imageWidth * imageHeight];

// Set up a pixel grabber to get the pixels
            PixelGrabber grabber = new PixelGrabber(
        coloringBook.getSource(),0, 0, imageWidth, imageHeight, imagePixels,0, imageWidth);

// Ask the image grabber to go get the pixels
            try {
        grabber.grabPixels();
            } catch (Exception e) {
        // Ignore for now
        return;
            }

// Make sure that the image copied correctly, although we don't
// do anything if it doesn't.

            if ((grabber.status() & ImageObserver.ABORT) != 0)
            {
        // uh oh, it aborted
        return;
            }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }

// getPixel returns the pixel value for a particular x and y
    private int getPixel(int x, int y)
        {
            return imagePixels[y * imageWidth + x];
        }

// setPixel sets the pixel value for a particular x and y
    private void setPixel(int x, int y, int color)
        {
            imagePixels[y * imageWidth + x] = color;
        }

// floodFill starts at a particular x and y coordinate and fills it, and all
// the surrounding pixels with a color. It doesn't paint over black pixels,
// so they represent the borders of the fill.
// The easiest way to code a flood fill is by doing it recursively - you
// call flood fill on a pixel, color that pixel, then it calls flood fill
// on each surrounding pixel and so on. Unfortunately, that usually causes
// stack overflows since recursion is pretty expensive.
// This routine uses an alternate method. It makes a queue of pixels that
// it still has to fill. It takes a pixel off the head of the queue and
// colors the pixels around it, then adds those pixels to the queue. In other
// words, a pixel is really added to the queue after it has been colored.
// If a pixel has already been colored, it is not added, so eventually, it
// works the queue down until it is empty.

        private void floodFill(int x, int y, int color)
        {
// If the pixel we are starting with is already black, we won't paint
            if (getPixel(x, y) == blackRGB)
            {
                return;
            }

// Create the pixel queue. Assume the worst case where every pixel in the
// image may be in the queue.
            int pixelQueue[] = new int[imageWidth * imageHeight];
            int pixelQueueSize = 0;

// Add the start pixel to the queue (we created a single array of ints,
// even though we are enqueuing two numbers. We put the y value in the
// upper 16 bits of the integer, and the x in the lower 16. This gives
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// a limit of 65536x65536 pixels, that should be enough.)

            pixelQueue[0] = (y << 16) + x;
            pixelQueueSize = 1;

// Color the start pixel.
            setPixel(x, y, color);
// Keep going while there are pixels in the queue.
            while (pixelQueueSize > 0)
            {

// Get the x and y values of the next pixel in the queue
                x = pixelQueue[0] & 0xffff;
                y = (pixelQueue[0] >> 16) & 0xffff;

// Remove the first pixel from the queue. Rather than move all the
// pixels in the queue, which would take forever, just take the one
// off the end and move it to the beginning (order doesn't matter here).

                pixelQueueSize--;
                pixelQueue[0] = pixelQueue[pixelQueueSize];

// If we aren't on the left side of the image, see if the pixel to the
// left has been painted. If not, paint it and add it to the queue.
                if (x > 0) {
                    if ((getPixel(x-1, y) != blackRGB) && (getPixel(x-1, y) != color)){
                        setPixel(x-1, y, color);
                        pixelQueue[pixelQueueSize] = (y << 16) + x-1;
                        pixelQueueSize++;
                    }
                }

// If we aren't on the top of the image, see if the pixel above
// this one has been painted. If not, paint it and add it to the queue.
                if (y > 0) {
                    if ((getPixel(x, y-1) != blackRGB) && (getPixel(x, y-1) != color))
                    {
                        setPixel(x, y-1, color);
                        pixelQueue[pixelQueueSize] = ((y-1) << 16) + x;
                        pixelQueueSize++;
                    }
                }

// If we aren't on the right side of the image, see if the pixel to the
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// right has been painted. If not, paint it and add it to the queue.
            if (x < imageWidth-1) {
                if ((getPixel(x+1, y) != blackRGB) &&
                    (getPixel(x+1, y) != color))
                {
                    setPixel(x+1, y, color);
                    pixelQueue[pixelQueueSize] = (y << 16) + x+1;
                    pixelQueueSize++;
                }
            }

// If we aren't on the bottom of the image, see if the pixel below
// this one has been painted. If not, paint it and add it to the queue.
            if (y < imageHeight-1) {
                if ((getPixel(x, y+1) != blackRGB) &&(getPixel(x, y+1) != color)){
                    setPixel(x, y+1, color);
                    pixelQueue[pixelQueueSize] = ((y+1) << 16) + x;
                    pixelQueueSize++;
                }
            }
        }

// remakeDisplayImage takes the array of pixels and turns it into an
// image for us to display.
    private void remakeDisplayImage()
    {
        displayImage = createImage(new MemoryImageSource(
            imageWidth, imageHeight, imagePixels, 0, imageWidth));
    }

// The paint method is written with the assumption that the screen has
// not been cleared ahead of time, that way we can create an update
// method that doesn't clear the screen, but doesn't need an off-screen
// image.

    public void paint(Graphics g)
    {
        int i;

// If we got the image successfully, draw it, otherwise, print a message
// saying we couldn't get it.

        if (imageValid)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            {
                g.drawImage(displayImage, 0, 0, this);
            }
            else
            {
                g.drawString("Unable to load coloring image.", 0, 50);
            }

// Draw the crayons
            for (i=0; i < crayons.length; i++)
            {
// Draw each crayon in the color it represents
                g.setColor(crayonColors[i]);
                g.fillPolygon(crayons[i]);

// Get the box that would enclose the crayon
                Rectangle box = crayons[i].getBoundingBox();
// If the crayon is the current one, draw a black box around it, if not,
// draw a box the color of the background around it (in case the current
// crayon has changed, we want to make sure the old box is erased).

                if (crayonColors[i] == currentDrawingColor)
                {
                    g.setColor(Color.black);
                }
                else
                {
                    g.setColor(getBackground());
                }

// Draw the box around the crayon.
                    g.drawRect(box.x, box.y, box.width, box.height);
            }
        }

// Override the update method to call paint without clearing the screen.

        public void update(Graphics g)
        {
            paint(g);
    }

    public boolean mouseDown(Event event, int x, int y)
    {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        int i;

    // Check each crayon to see whether the mouse was clicked inside of it. If so,
    // change the current color to that crayon't color. We use the "inside"
    // method to see whether the mouse x,y is within the crayon shape. Pretty handy!

        for (i=0; i < crayons.length; i++)
        {
            if (crayons[i].inside(x, y))
            {
                currentDrawingColor = crayonColors[i];
                repaint();
                return true;
            }
        }

    // If the mouse wasn't clicked on a crayon, see whether it was clicked within
    // the image. This assumes that the image starts at 0, 0.
        if ((x < imageWidth) && (y < imageHeight))
        {
    // If the image was clicked, fill that section of the image with the
    // current crayon color
            floodFill(x, y, currentDrawingColor.getRGB());

    // Now re-create the display image because we just changed the pixels
            remakeDisplayImage();
            repaint();
            return true;
        }

        return true;
    }
}
```

25.6 颜色模式

图像生产者-消费者模型还使用了 ColorModel 类。用户已经看到了在生产者和消费者之间传递的图像是由整数数组完成的。每个整数代表一个像素的颜色。ColorModel 类包含的方法可以从像素值中抽取红色分量、绿色分量、蓝色分量和阿尔法分量。用户可能已经熟悉了红、绿和蓝色分量，但是阿尔法分量可能对用户来说有些陌生。

参见第 22 章的“Color 类”部分

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

阿尔法分量代表着某个颜色的透明度。阿尔法值为 255 意味着该颜色完全不透明，而阿尔法值为零则表明该颜色是完全透明的。缺省的颜色模式是 RGBdefault 模式，该模式将四色分量编码为 0xaarrggbb 格式。最左边八位是阿尔法值，接下来八位是红色分量，再接下来的是绿色分量，最后八位是蓝色分量。例如某个颜色 0x12345678 拥有阿尔法分量 0x12（透明度较好）、红色分量 0x34、绿色分量 0x56 和蓝色分量 0x78。

注意：阿尔法分量只适用于图像。用户不能把它连同 Color 类一起使用。换句话说，用户不能在 Graphics 类的任何绘画函数中使用它。

在需要颜色模式并且使用 RGBdefault 就可以满足需要时，用户可以使用 getRGBdefault 获得当前的颜色模式：

```
public static ColorModel getRGBdefault()
```

用户可以使用下列方法抽取某个像素的红色分量、绿色分量、蓝色分量和阿尔法分量：

```
public abstract int getRed(int pixel)
public abstract int getGreen(int pixel)
public abstract int getBlue(int pixel)
public abstract int getAlpha(int pixel)
```

用户可以使用 getPixelsize 方法求出每个像素的位数：

```
public int getPixelSize()
```

因为许多 AWT 部件使用 RGB 格式的颜色，所以用户可以使用 getRGB 方法请求颜色模式把像素值转换成 RGB 格式：

```
public int getRGB(int pixel)
```

25.6.1　DirectColorModel 类

DirectColorModel 类将红色、绿色、蓝色和阿尔法分量直接存储在像素值中。标准的 RGB 格式是一个直接颜色模式的例子。该像素格式由一套掩码决定，该掩码告诉颜色模式如何把每种颜色映射到像素中。DirectColorModel 的构造器使用每个像素的位数、红色、绿色和蓝色位掩码以及可选的阿尔法掩码作为参数：

```
public DirectColorModel(int bits,int redMask,int greenMask,int blueMask)
public DirectColorModel(int bits,int redMask,int greenMask,int blueMask,int alphaMask)
```

用户可以使用下列方法查询掩码值：

```
public final int getRedMask()
public final int getGreenMask()
public final int getBlueMask()
public final int getAlphaMask()
```

每个掩码的位必须是接触着的，即它们全部是毗连的。用户不能把蓝色位放在两个红色位中间。标准的 RGB 格式是 0xaarrggbb，其中 aa 是阿尔法分量的十六进制值，而 rr、gg、bb 分别代表红色、绿色和蓝色分量的十六进制值。在直接颜色模式中应该表示为：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
DirectColorModel rgbModel = new DirectColorModel(32,0xff0000,0x00ff00,0x0000ff,0xff000000)
```

25.6.2 IndexColorModel 类

与 DirectColorModel 不同，IndexColorModel 类将实际的红色、绿色、蓝色和阿尔法分量存放在像素之外。颜色值是一个指向颜色表的索引。用户可以通过向构造器传递每个像素的位数、颜色表中的项数，红色、绿色和蓝色分量来创建 IndexColorModel。用户可以有选择地为透明像素传递阿尔法分量或索引值：

```
public IndexColorModel(int bitsPerPixel,int tableSize,byte[] red,byte[] green,byte[] blue)
public IndexColirModel(int pitsPerPixel,int tableSize,byte[] red,byte[] green,byte[] blue,int transparent-
Pixel)
public IndexColorMldel(int bitsperpixel,int table size ,byte[] red, byte[] gereen,byte[] blue,byte[] alpha)
```

用户不必分别传递红、绿和蓝色分量数组，只要将它们作为一个大的字节数组传递就可以。IndexColorModel 类假设每三个字节代表一种颜色（如果用户要发送阿尔法分量，那么就是每四个字节代表一种颜色）。颜色分量应该按照红色、绿色和蓝色的顺序存放。如果指定了阿尔法分量，那么阿尔法分量应该排列在蓝色后面。也可能与直觉相反的顺序，因为标准的 RGB 格式首先是阿尔法分量。下面是各种颜色打包格式的构造器：

```
pablic IndexColorModel(int bitsPerPixel,int tableSize,byte[] packedTable,boolean incl udeAlpha)
public IndexColorModel(int bitsPerPixel,int tableSize,byte[] packedTable,boolean includeAlpha,int transpar-
entPixel)
```

注意使用最后这种格式时，用户实际上同时拥有透明像素和阿尔法分量！

使用下面的方法用户可以获得红色、绿色、蓝色和阿尔法表的拷贝。

```
public final void getReds(byte[] redArray)
public final void getGreens(byte[] greenArray)
public findl void getBlues(byte[ | blueArray)
public finel void getAlphas(byte[] alphaArray)
```

每个方法将分量值从表中拷贝到用户传递给方法的数组中。要确保该数组的尺寸至少与表的尺寸一样大。getMapSize 方法返回该表的尺寸：

```
public final int getMapSize()
```

getTransparentPixel 方法返回透明像素的索引值，如果没有透明像素，那么该方法返回 -1：

```
public final int getTransparentPixel()
```

25.6.3 RGBImageFilter 类

java.awt.image 包提供了两个标准的图像过滤器：CropImageFilter 和 RGBImageFilter。RGBImageFilter 允许用户在不改变图像本身的前提下操作图像的颜色。当用户创建自己的自定义 RGBImageFilter 时，用户只需要创建 filterRGB 方法即可：

```
public abstract int filterRGB(int x,int y,int rgb)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

对于图像的每个像素，filterRGB 方法需要传递该像素的 x 坐标和 y 坐标及它当前的 RGB 值。该方法的返回值为该像素新的 RGB 值。

因为某些图像是被用索引颜色模式定义的，所以用户可以设置自己的过滤器，该过滤器只能过滤索引颜色模型。如果颜色调整与像素的 x 坐标和 y 坐标值无关，那么这种过滤器是十分方便的。如果用户只过滤来自索引的 RGB 值，传递给 filterRGB 的 x 和 y 坐标值将为 -1，-1。为了指明用户希望过滤该索引而不是整个图像，用户可以将变量 canFilterIndexColorModel 的值设置为 true：

```
protected boolean canFilterIndexColorModel
```

如果用户希望改变索引颜色模式的过滤行为，那么用户可以覆盖 filterIndexColorModel 方法：

```
public indexColorModel filterIndexColorModel (IndexColormodel oldCM)
```

由该方法返回的 IndexColorModel 是该图像将要使用的新索引颜色模式。

如果用户希望只为某个图像改变颜色模式，那么用户可以使用 RGBImageFilter 将一种颜色模式替换成另一种：

```
public void substituteColorModel(ColorModel oldCM, ColorModel newCM)
```

当过滤索引颜色模型时该方法由 RGBImageFilter 使用。该方法使用用户的 filterRGB 方法过滤旧模式的颜色来创建一个新的颜色模式，然后建立一个从旧颜色模式到新颜色模式的替换。当替换被设置后，filterRGB 方法就不再为单独的像素而被调用了。这种方式允许用户快速地改变颜色。

程序清单 25.7 显示了一个简单的灰度颜色模式类，该类从别的颜色模式中取得红色、绿色和蓝色值并将这些颜色值转换或灰度值。它取出最大的红色分量、绿色分量和蓝色分量并把它作为所有像素的三个分量。灰度颜色模型没有使用阿尔法值。

```
import java.awt.image.*;

// This class implements a gray color model
// scheme based on another color model. It acts
// like a gray filter. To compute the amount of
// gray for a pixel, it takes the max of the red,
// green, and blue components and uses that value
// for all three color components.

public class GrayModel extends ColorModel
{
   ColorModel originalModel;

   public GrayModel(ColorModel originalModel)
   {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        super(originalModel.getPixelSize());
        this.originalModel = originalModel;
    }

// The amount of gray is the max of the red, green, and blue
    protected int getGrayLevel(int pixel)
    {
        return Math.max(originalModel.getRed(pixel),
            Math.max(originalModel.getGreen(pixel),
                originalModel.getBlue(pixel)));
    }

// Leave the alpha values untouched
    public int getAlpha(int pixel)
    {
        return originalModel.getAlpha(pixel);
    }

// Since gray requires red, green and blue to be the same,
// use the same gray level value for red, green, and blue

    public int getRed(int pixel)
    {
        return getGrayLevel(pixel);
    }

    public int getGreen(int pixel)
    {
        return getGrayLevel(pixel);
    }

    public int getBlue(int pixel)
    {
        return getGrayLevel(pixel);
    }

// Normally, this method queries the red, green, blue and
// alpha values and returns them in the form 0xaarrggbb. To
// keep from computing the gray level 3 times, we just override
// this method, get the gray level once, and return it as the
// red, green, and blue, and add in the original alpha value.

    public int getRGB(int pixel)
    {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        int gray = getGrayLevel(pixel);
        return (getAlpha(pixel) << 24) + (gray << 16) +
           (gray << 8) + gray;
    }
}
```

程序清单25.8显示了一个RGB图像过滤器,它将原来的颜色模式替换成灰度模式。

```
import java.awt.image.*;

// This class sets up a very simple image graying
// filter. It takes the original color model and
// sets up a substitition to a GrayModel.
public class GrayFilter extends RGBImageFilter
{
    public GrayFilter()
    {
        canFilterIndexColorModel = true;
    }

// When the color model is first set, create a gray
// model based on the original model and set it up as
// the substitute color model.

    public void setColorModel(ColorModel cm)
    {
        substituteColorModel(cm, new GrayModel(cm));
    }

// This method has to be present, but it will never be called
// because we are doing a color model substitution.

    public int filterRGB(int x, int y, int pixel)
    {
        return pixel;
    }
}
```

程序清单25.9显示了一个简单的小应用程序,该小应用程序使用灰度过滤器显示了一幅图像。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.awt.*;
import java.awt.image.*;
import java.applet.*;

// This applet displays a grayed-out image by using
// a GrayFilter rgb image filter.

public class Grayer extends Applet
{
    private Image origImage;
    private Image grayImage;
    private GrayFilter colorFilter;

    public synchronized void init()
    {
// Get the name of the image to use
        String gifName = getParameter("image");

// Fetch the image
        origImage = getImage(getDocumentBase(), gifName);
        System.out.println(origImage);

// Create the gray filter
        colorFilter = new GrayFilter();

// Create a grayed-out version of the original image
        grayImage = createImage(new FilteredImageSource(
            origImage.getSource(),
            colorFilter));

        MediaTracker mt = new MediaTracker(this);
        mt.addImage(grayImage, 0);
        try {
            mt.waitForAll();
        } catch (Exception ignore) {
        }

    }

    public synchronized void paint(Graphics g)
    {
        g.drawImage(grayImage, 0, 0, this);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }

    public void update(Graphics g)
    {
        paint(g);
    }
}
```

25.6.4　通过颜色循环产生动画

颜色循环技术是许多人都知道的一种动画技术，图像的动画效果是由改变调色板产生的，并未改变实际的图像。这种技术可以采用多种形式——从模拟流水到改变文本。用户可以把这种技术使用在索引颜色模式上。动画思想就是改变颜色表的值并用新的颜色表重画图像。如果用户继续循环一套颜色，那么尽管图像数据本身并不改变，但是图像都可以产生动画效果。

提示：当用户执行通过在空中创建新图像(on-the-fly)产生的图像动画时，不要使用 CreateImage 方法去产生新图像，应该调用当前图像的 flush 方法重用已有图像。这可以清除由老图像使用的内存并使老图像被重新过滤。否则，在某些系统上用户可能会消耗更多的内存。

程序清单 25.10 显示了一个 RCB 图像过滤器，它循环索引颜色模式中的颜色。

```
import java.awt.*;
import java.awt.image.*;

//
// This class cycles the colors in an index color model.
// When you create a CycleFilter, you give the offset in
// the index color model and also the number of positions
// you want to cycle. Then every time you call cycleColors,
// it increments the cycle position. You then need to re-create
// your image and its colors will be cycled.
//
// This filter will work only on images that have an indexed
// color model.
public class CycleFilter extends RGBImageFilter {

  // The offset in the index to begin cycling
  protected int cycleStart;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    // How many colors to cycle
    protected int cycleLen;

    // The current position in the cycle
    protected int cyclePos;

    // A temporary copy of the color components being cycled
    protected byte[] tempComp;

    public CycleFilter(int cycleStart, int cycleLen) {
      this.cycleStart = cycleStart;
      this.cycleLen = cycleLen;
      tempComp = new byte[cycleLen];

      cyclePos = 0;

      // Must set this to true to allow the shortcut of filtering
      // only the index and not each individual pixel

      canFilterIndexColorModel = true;
    }

    // cycleColorComponent takes an array of bytes that represent
    // either the red, green, blue, or alpha components from the
    // index color model, and cycles them based on the cyclePos.
    // It leaves the components that aren't part of the cycle intact.

    public void cycleColorComponent(byte component[]) {

      // If there aren't enough components to cycle, leave this alone
      if (component.length < cycleStart + cycleLen) return;

      // Make a temporary copy of the section to be cycled
      System.arraycopy(component, cycleStart, tempComp,
                       0, cycleLen);

      // Now for each position being cycled, shift the component over
      // by cyclePos positions.
      for (int i = 0; i < cycleLen; i++) {
        component[cycleStart + i] = tempComp[(cyclePos + i) % cycleLen];
      }
    }

    // cycleColors moves the cyclePos up by 1.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public void cycleColors() {
  cyclePos = (cyclePos + 1) % cycleLen;
}

// Can't really filter direct color model RGB this way, since we have
// no idea what rgb values get cycled, so just return the original
// rgb values.

public int filterRGB(int x, int y, int rgb) {
  return rgb;
}

// filterIndexColorModel is called by the image filtering mechanism
// whenever the image uses an indexed color model and the
// canFilterIndexColorModel flag is set to true. This allows you
// to filter colors without filtering each and every pixel
// in the image.

public IndexColorModel filterIndexColorModel(IndexColorModel icm) {

  // Get the size of the index color model
  int mapSize = icm.getMapSize();

  // Create space for the red, green, and blue components
  byte reds[] = new byte[mapSize];
  byte greens[] = new byte[mapSize];
  byte blues[] = new byte[mapSize];

  // Copy in the red components and cycle them
  icm.getReds(reds);
  cycleColorComponent(reds);

  // Copy in the green components and cycle them
  icm.getGreens(greens);
  cycleColorComponent(greens);

  // Copy in the blue components and cycle them
  icm.getBlues(blues);
  cycleColorComponent(blues);

  // See if there is a transparent pixel. If not, copy in the alpha
  // values, just in case the image should be partially transparent.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    if (icm.getTransparentPixel() == -1) {

      // Copy in the alpha components and cycle them
      byte alphas[] = new byte[mapSize];
      icm.getAlphas(alphas);
      cycleColorComponent(alphas);

      return new IndexColorModel(icm.getPixelSize(),
                                 mapSize, reds, greens, blues, alphas);
    } else {

      // If there was a transparent pixel, ignore the alpha values and
      // set the transparent pixel in the new filter
      return new IndexColorModel(icm.getPixelSize(),
                                 mapSize, reds, greens, blues,
                                 icm.getTransparentPixel());
    }
  }
}
```

为了使用 CycleFilter,用户可以编写一个小应用程序,该小应用程序连续地在 CycleFilter 中调用 cycleColors 然后重画图像。程序清单 25.11 显示了一个小应用程序例子,它使用索引颜色模式创建了一个简单的内存图像,并使用 CycleFilter 循环颜色。图 25.11 显示了由 Cycler 小应用程序产生的输出图像。

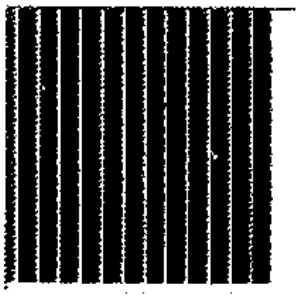


图 25.11 Cycler 小应用程序通过循环调色板产生动画效果

```
import java.awt.*;
import java.awt.image.*;
import java.applet.*;

// This applet creates a series of moving
// lines by creating a memory image and cycling
// its color palette.

public class Cycler extends Applet implements Runnable {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
protected Image origImage; // the image before color cycling
protected Image cycledImage;   // image after cycling
protected CycleFilter colorFilter;   // performs the cycling

protected Thread cycleThread;
protected int delay = 50;   // milliseconds between cycles

protected int imageWidth = 200;
protected int imageHeight = 200;

protected boolean stopStatus = false; //thread should not stop, until true

public void init() {

  // Create space for the memory image
  byte pixels[] = new byte[imageWidth * imageHeight];

  // We're going to cycle through 16 colors, but leave position 0 alone in
  // the index color model we create, so allow room for 17 slots
  byte red[] = new byte[17];
  byte green[] = new byte[17];
  byte blue[] = new byte[17];

  // Fill slots 1-16 with varying shades of gray (when the red, green,
  // blue values are all equal you get shades of gray ranging from
  // black when all values are 0, to white when all values are 255).

  for (int i=0; i < 16; i++) {
    red[i+1] = (byte) (i * 16);
    green[i+1] = (byte) (i * 16);
    blue[i+1] = (byte) (i * 16);
  }

  // Create an index color model that supports 8 bit indices, only 17
  // colors, and uses the red, green, and blue arrays for the color values

  IndexColorModel colorModel = new IndexColorModel(8, 17,red, green, blue);

  // Now create the image, just go from top to bottom, left to right
  // filling in the colors from 1-16 and repeating.

  for (int i=0; i < imageHeight; i++) {
    for (int j=0; j < imageWidth; j++) {
      pixels[i * imageWidth + j] = (byte) ((j % 16) + 1);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            }
        }

        // Create the uncycled image
        origImage = createImage(new MemoryImageSource(imageWidth, imageHeight, colorModel, pixels, 0,
imageWidth));

        // Create the filter for cycling the colors
        colorFilter = new CycleFilter(1, 16);

        // Create the first cycled image
        cycledImage = createImage(new FilteredImageSource(origImage.getSource(),colorFilter));
    }

    // Paint simply draws the cycled image
    public synchronized void paint(Graphics g) {
        g.drawImage(cycledImage, 0, 0, this);
    }

    // Flicker-free update
    public void update(Graphics g) {
        paint(g);
    }

    // Cycles the colors and creates a new cycled image. Uses media
    // tracker to ensure that the new image has been created before
    // trying to display. Otherwise, we can get bad flicker.

    public synchronized void doCycle() {
        // Cycle the colors
        colorFilter.cycleColors();
        // Flush clears out a loaded image without having to create a
        // whole new one. When we use waitForID on this image now, it
        // will be regenerated.

        cycledImage.flush();

        MediaTracker myTracker = new MediaTracker(this);
        myTracker.addImage(cycledImage, 0);
        try {

            // Cause the cycledImage to be regenerated
            if (! myTracker.waitForID(0, 1000)) {
                return;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
      } catch (Exception ignore) {
      }
      // Now that we have reloaded the cycled image, ask that it
      // be redrawn.
      repaint();
    }

    // Typical threaded applet start and stop
    public void start() {
      stopStatus = false; //don't stop yet
      cycleThread = new Thread(this);
      cycleThread.start();
    }

    public void stop() {
      stopStatus = true;
    }

    public void run() {
      // Continually cycle colors and wait.
      while (! stopStatus) {
        doCycle();
        try {
          Thread.sleep(delay);
        } catch (Exception hell) {
        }
      }
    }
  }
```

当用户熟悉了 Java 的图像模式时，用户可以创建许多奇妙的图像。用户可以通过编写图像过滤器来产生各种不同的效果，可以使用 MemoryImageSource 和 PixelGrabber 建造图像编辑器或编写绘画程序，甚至可以利用图像透明获得有趣的图像组合。只要是用户需要的图像操作，Java 都能够处理。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 26 章　Java 二维图形

本章内容：

26.1　Graphics2D 对象
26.2　Java 二维应用编程接口中的坐标
26.3　画图
26.4　不同的笔划
26.5　自定义填充
26.6　转换
26.7　绘制文本
26.8　绘制图像
26.9　透明
26.10　裁剪

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

26.1　Graphics2D 对象

Java 早期版本的主要缺陷之一是它的图形应用程序编程接口不够健壮。由 AWT 提供的这些功能粗糙得与八十年代早期八位计算机提供的画图功能差不多。Java1.2 引进了新的二维图形应用编程接口，它提供的现代图形应用编程接口能够提供的各种特性。Java 二维应用编程接口提供了较好的对绘制形状、填充形状、旋转形状、绘制文本、绘制图像和定义颜色支持。

如果用户已经从 Java AWT 中熟悉了 Graphics 对象，那么用户可能会有一些二维图形应用编程接口工作原理的概念。所有新增的二维特性都是由 Graphics2D 类提供的，该类是 Graphics 的子类。当 Java 设计者创建新的二维应用编程接口时面临着挑战，因为许多开发人员都已经熟悉了已有的图形应用编程接口，他们需要继续扩展老的应用编程接口，因为新的 Graphics2D 对象是 Graphics 的子类，所以用户可以在任何使用 Graphics 对象的地方使用 Graphics2D 对象。换句话说用户已有的图形代码在使用新的 Graphics2D 对象时不必中止。

在老的 AWT 应用编程接口中，用户并不实际支持创建自己的 Graphics 对象（至少不是通常）。用户依靠的是 paint 方法，paint 方法接收 Graphics 对象作为参数。在二维应用编程接口中，用户仍然使用 paint 方法去获得 Graphics 对象。在 Java1.2 中传递给 paint 方法的 Graphics 对象实际是一个 Graphics2D 对象！用户只需要将 Graphics 对象强制成 Graphics2D 对象就可以：

```
public void paint (Graphics oldGraphics)

    Graphics2D newGraphics = (Graphics2D)oldGraphics;
    //现在用户就可以使用新的二维方法了
}
```

26.2　Java 二维应用编程接口中的坐标

原来的 AWT 应用编程接口将绘画区看作是简单的像素域。坐标 1,1 代表一个像素位置，而坐标 2,1 是与 1,1 直接毗邻的像素。这种绘画坐标系统只用于简单的屏幕绘画，当用户试图在打印机上绘画时这种绘画坐标系统就难以很好地工作了。问题在于打印机拥有比屏幕更大的分辨率。如果用户使用屏幕上图像的点阵数在打印机上绘制同一个图像，那么打印机上的图像将是非常小的。（或者非常粗糙的）。作为开发者，用户并不希望具体操作对各种设备的缩放处理，所以应该由二维应用编程接口负责处理这方面工作。

Java 二维应用编程接口有一个“坐标空间”概念。坐标空间就是指确定坐标的方式。用户的程序通常运行在“用户坐标空间”中，这是一个虚拟的绘画区域，它的坐标点 0,0 在左上角并且该区域有一定的像素宽度和一定的像素高度。因为大部分绘画工作是在屏幕上进行的，所以给出的用户坐标空间与屏幕区域有相同的尺寸。当用户的程序运行在用户坐标空间时，该设备使用“设备坐标空间”显示该输出。为了实现用户坐标和设备坐标的转换，二维系统使用了“缺省转换”，该转换定义了如何将用户空间的坐标转换成设备空间的坐标。因为用户空间的尺寸与屏幕尺寸相同，所以屏幕设备的缺省转换根本不改变坐标。换句话说，用户空间的坐

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

标 100,50 也是屏幕的坐标 100,50。打印机缺省的转换在一英寸的位置上放置 72 个用户空间像素。打印页上用户空间的坐标点 144,72 位于距左边两寸,距顶边一寸的位置。

虽然使用用户空间的坐标代表像素有时是正确的,但它并不总是正确的。在 Graphics2D 中坐标全部是用浮点数表示的,记住不是用整数描述。所以虽然用户可以绘制一条从 50.1,100.1 到 50.4,100.4 的直线,但是在屏幕上实际上只是在 50,100 处画了一个点;而在打印机上用户将会得到非常短的直线。

这种坐标不必代表全部像素的事实允许用户制作非常光滑的打印机图片和在屏幕上执行复杂的图像转换。用户现在可以以很高的精度绘制一幅图像,然后让屏幕设备尽可能好地将它绘制出来。如果用户拥有更好的显示设备,那么图画看上去会更漂亮些。在过去因为用户是工作在全部像素上面,所以打印设备无法给出更精美的细节,因为无法更加具体。如果用户想在屏幕上画一个直径为三个像素的圆,那么用户要么画出一个加号(+),要么就画一个3×3的方块。使用 Java 二维应用编程接口,用户画出的直径为 3 的圆在屏幕上仍然看上去或者像方块或者像加号,但是在打印机上可是一个不折不扣的圆。

当用户编写 Java 程序时不必直接处理坐标空间,但是记住使用 Graphics2D 对象可以比使用老的 Graphics 对象更详细地指定坐标是非常重要的。

26.3 画图

在原来的 AWT 的 Graphics 对象中有绘制和填充各种形状的特殊方法。但是几乎没有选项来指定绘制和填充形状的笔。事实上,用户只有一个可以改变的东西就是用于绘制和填充的颜色。

Graphics2D 对象将绘制好的图当做形状来对待(它分别对待图像和文本)。在 Graphics2D 中只有一个 draw 方法用于绘制,它使用形状作为参数:

```
public void draw(Shape s)
```

类似地,Graphics2D 也只有一个填充方法:

```
public void fill(Shape s)
```

在二维应用编程接口中画图的技巧在于创建形状。幸运的是 java.awt.geom 包提供了各种形状创建类,包括一些常用的形状。

提示:当使用形状对像时不要忘记引入 java.awt.geom.*。只引入 java.awt.*是不能引入 java.awt.geom.*子包的。

26.3.1 画线

java.awt.geom.Line2D 类是描述两点间简单直线形状的。当用户创建直线时,用户必须选用 line 类的 Float 版本和 Double 版本之一来定义点。Double 版本在需要较高精度时使用。下列的代码片段创建了一个从(50,60)到(300,320)的 Line2D 对象并将它们绘制出来:

```
Line2D line = new Line2D.Float(50, 60, 300, 320);
newGraphics.draw(line);
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

注意：在 java.awt.geom 包中，用户可以发现 Java 的设计者大量地使用了 Java 的内置类。所有的形状对象都有至少一个内置类，这些类被用于使用特殊数据类型来指定坐标。所有的形状都有一个 Float 型内置类，并且大部分形状都有一个 Double 类。Line2D 是一个形状类，但是为了指定坐标，用户需要使用内置类 Line2D.Float。

用户也可以使用 Point2D 类来定义某个直线的端点：

```
Line2D line = new Line2D.Float (new Point2D.Float(50, 60),new Point2D.Float(300, 320));
```

26.3.2　绘制矩形

Rectangle2D 类使用一个 x，y 坐标、一个宽度和一个高度定义一个矩形形状。下面的代码片段在(10,15)处创建一个宽为 100 高为 50 的矩形并将该矩形绘制出来：

```
Rectangle2D rect = new Rectangle2D.Float(10, 15, 100, 50);
newGraphics.draw(rect);
```

26.3.3　绘制圆角矩形

RoundRectangle2D 画出的矩形的角是圆的。除了矩形的位置和尺寸外，用户可以指定角部圆弧的宽度和高度。如果用户为该圆弧指定了 5 个单位的宽度和高度，那么在每个角上的圆形部分都具有 5 个单位的宽度和高度。下面的代码片段在 10,15 处定义了一个宽为 100 高为 50 并且带有 x 方向上 5 个单位 y 方向 3 个单位圆角的矩形：

```
RoundRectangle2D roundRect = newRoundRectangle2D.Float(10,15,100, 50, 5, 3);
```

26.3.4　绘制椭圆和圆

Ellipse2D 描述一个椭圆形状，当宽度和高度相等时该形状就是一个圆。与在 Graphics 对象中使用 ellipse 方法一样，椭圆也是使用外接矩形定义的，用户需要指定一个矩形，该矩形的宽度和高度分别代表椭圆的宽度和高度(对于圆形，可以使用圆的直径作为宽度和高度)。用户还需要给出该矩形左上角的坐标。下面的代码片段在坐标 50,80 处创建一个宽为 30 高为 10 的椭圆：

```
Ellipse2D ellipse = new Ellipse2D.Float(50, 80, 30, 10);
```

提示：用户经常使用形状的中心来指定圆或椭圆的坐标，而不使用外接矩形左上角的坐标来指定圆或椭圆的坐标。用户可以使用公式 leftx = centerX - (width/2)和 upperY = centerY - (height/2)计算出左上角坐标。

26.3.5　画弧

除了绘制椭圆以外，用户可以使用 Arc2D 类绘制部分椭圆弧，用户可以如同指定椭圆一样，使用外接矩形的左上角、宽度和高度指定弧的位置和大小。另外用户还需要以度为单位指定起始角和终止角。对于半个椭圆，用户可以指定起始角为 0 度，终止角为 180 度，或者起始

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

角为 90 度和终止角为 270 度等。记住和罗盘不同,0 度点在右边,90 度指向下方、180 度指向左和 270 度指向上方。

警告:Arc2D 类的角度是沿顺时针方向增加的(90 度指向下方)。这与 Grtaphics 类中老的弧过程是相反的,Graphics 类中的 90 度位于上方并且角度是按逆时针方向增加的。

另外用于指定弧的起点和终点的角度可能看上去与用户期望的并不一致。角度是从椭圆的中心开始测量的,但是它们是按照圆的角度测量的。如果椭圆的宽度是高度的两倍,那么从屏幕上测量出的角度与用户指定的角度是不一致的。图 26.1 显示了一个起始角为 0 度的 45 度扇状楔形和用椭圆画来的楔形。注意屏幕上第二个楔形的角度小于 45 度。



图 26.1 当圆伸展成椭圆时,弧的角度在屏幕上显示的角度与屏幕上显示的角度并不相配

26.3.6 画曲线

除了弧线以外,Java 二维应用编程接口提供了由二次曲线和三次曲线定义的形状。二次曲线是由两个端点和一个控制曲线开头的单独控制点定义的。三次曲线类似于二次曲线,所不同的是它有两个控制点而不是一个控制点。

下面的代码片段使用端点 50,50 和 50,200 及控制点 10,10 定义了一条二次曲线:

```
QuadCurve2D curve = new QuadCurve2D.Float(50,50,10,10, 50,100);
```

图 26.2 显示了一条二次曲线并在控制点处画了一个圆。

下面的代码片段使用端点 50,50 和 50,200 及控制点 10,10 和 100,100 定义了一条三次曲线:

```
CubicCurve2D curve = new CubicCurve2D.Float(50, 50, 10, 10, 100, 100, 50, 200);
```

图 26.3 显示了一条三次曲线并在控制点处各画了一个圆。



图 26.2 二次曲线加从于控制点

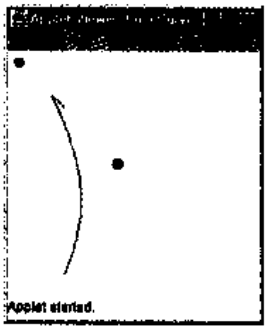


图 26.3 三次曲线屈从于两个控制点

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

提示：在所有的曲线定义函数中，控制点参数出现在起点和终点之间。换句话说前两个参数定义了起始点，(如果需要的话)接下来的几对参数定义的是控制点，最后两个参数定义的是终点。

26.3.7　绘制任意形状

除了预定义形状以外，用户也可以使用 GeneralPath 类创建用户自己的形状。创建自己的形状所要做的工作是对该形状的每个点调用 GeneralPath 的 moveTo 方法和 lineTo 方法，就好像使用 Graphics 类绘制该形状一样。

下面的代码片段创建了一个 GeneralPath 对象并使用 moveTo 和 lineTo 方法定义一个三角形：

```
GeneralPath path = new GeneralPath( );
path.moveTo(0, 50);
path.lineTo(25,0);
path.lineTo(50,50);
```

除了 lineTo 方法以外，用户还可以使用 quadTo 方法和 curveTo 方法在两个点之间创建一条曲线。quadTo 方法使用一个控制点从当前点开始向着另一个点画出一条二次曲线。下面的代码片段以 100,0 为控制点向 50,50 定义出一条二次曲线：

```
path.quadTo(100, 0, 50, 50);
```

curveTo 方法使用两个控制点，如下面的语句所示，除了额外增加一个控制点 40,30 以外与前面的 quadTo 语句一样：

```
path.curveTo(40, 30, 100, 0, 50, 50);
```

26.4　不同的笔划

直到目前为止，绘制各种形状的方式与原来老的绘制方式没有什么太大的差别。真正能够令人兴奋的是修改绘画的笔划和为用户绘制的形状填充图案。绘画的笔划定义了如何绘制形状的边。在老 AWT 的 Grapahics 对象中，笔划总是一个像素宽的实线。现在用户可以改变笔划的宽度，并且可以创建多种点线。

BasicStroke 类可以按各种方式创建对象：

```
public BasicStroke( )
public BasicStroke(float width,int cap,int join)
public BasicStroke(float width,int cap,int join,float miterlimit)
public BasicStroke(float width,int cap,int join,float miterlimit,float dash_phasee)
```

其中 width 参数定义笔划的宽度。缺省的宽度为 1。cap 参数决定笔划端部的形状。对于与周围形状没有连接关系的直线和曲线该参数是很重要的。cap 参数有三种可能的值：CAP_BUTT；CAP_ROUND 和 CAP_SQUARE。

CAP_BUTT 值指示在笔划的端部不再绘制任何东西。CAP_ROUND 指示该线的端头

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

是圆的，而 CAP_SQUARE 指示端头是方形的。图 26.4 显示了各种笔划端头。

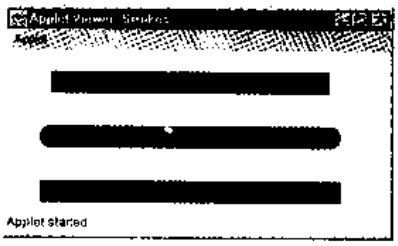


图 26.4 笔划可以是不加修饰的、圆头的和方头的

join 参数决定笔划中的角应该如何处理。join 参数有三个可能的值：JOIN_BEVE、JOIN_MITER 和 JOIN_ROUND。

JOIN_BEVEL 选项在笔划的外端画一条直线并填充该区域，这种角有着展平的外观。JOIN_MITER 选项延展笔划直到它们相交，在交汇处形成一个尖角。JOIN_ROUND 选项使得笔划端头产生一个弧线连接，在角上给出一个圆形外观。miterlimit 参数定义了斜角连接的长度。对于 JOIN_ROUND 和 JOIN_BEVE 它没有用途。图 26.5 显示了各种类型的角部连接。

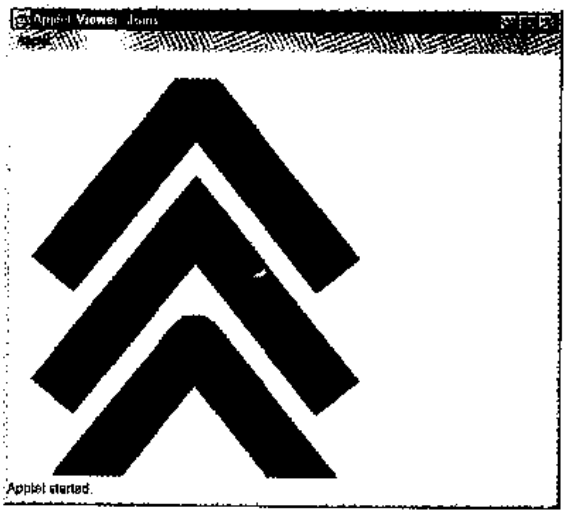


图 26.5 角部可以是倒角连接、斜角连接和圆角连接

dash 数组包含了空白和划线的交替长度。该数组的第一个元素指示第一个划线的长度。第二个元素指示了第一个空隙的长度。该数组后续的值继续在划线长度与间隙长度之间交替。例如，下面的数组定义了通过指定一个长的划线，一个小的间隙，一个短的划线和另一个与第一个间隙长度相同的间隙来创建一个点划线图案：

```
float[ ] dashValues = new float[4];
dashValue[0] = 20;
dashValue[1] = 10;
dashValue[2] = 5;
dashValue[3] = 10;
```

图 26.6 显示了这个点划线图案。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

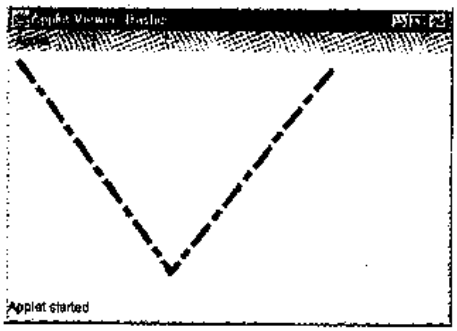


图 26.6　用户可以使用 Stroke 对象创建有趣的点划线图案

dashphase 参数讲述从划线序列何处开始。如果用户使用 10 作为第一个划线的长度，并将划线相位设置为 11，那么该线将从第一个空格开始。

当用户创建完一个 BasicStroke 对象以后，就可以通过调用 setStroke 方法让 Graphics2D 类使用它：

```
newGraphics.setStroke(myStroke);
```

26.5　自定义填充

原来的 AWT Graphics 类只允许单一颜色填充。Graphics2D 类增加了创建梯度颜色填充和纹理图案填充的能力。GradientPaint 对象不需要包含在用户正在画的图中，这些点只是用于按用户空间定义梯度区的长度。如果用户定义的起点和终点距离 100 个单位，那么填充颜色将在这 100 个像素距离上从起点的颜色逐步变为终点的颜色。正常情况下，渐变是非循环的，所以当颜色渐变到终点时颜色值就停留在终点颜色上。用户可以选择填充循环，这将导致填充颜色在到达终点后变回起点颜色。渐变方式沿用用户定义的两点之间连线上的渐变方式。换句话说，如果用户定义的点是在水平线上的，那么渐变将沿水平方向改变颜色。如果点在垂直线上，那么颜色将沿垂直方向渐变。如果点是在对角线上，那么渐变就沿对角线方向改变颜色。

图 26.7 显示了三种渐变图案和指示渐变方向的直线。

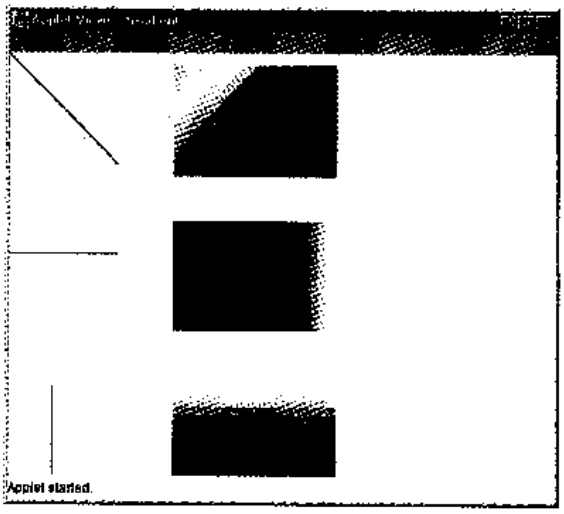


图 26.7　渐变沿着端点连线方向改变颜色

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

渐变听起来很复杂，但是实际上它是很容易设置的。GrandientPaint 对象拥有下列构造器：

```
public GradientPaint(float x1,float y1,Color color1,float x2,float y2,Color color2)
public GradientPaint(float x1,float y1,Color color1,float x2,float y2,Color color2,boolean cyclic)
public GradienPaint(PointID PT1,Color color1,Point2D pt2,Color color2)
public GradientPaint(PointID pt1,Color color1,Point2D pt2,Color color2,boolean cyclic)
```

正如用户看到的那样，借助于各种构造器，用户可以使用浮点 x，y 值或者 Point2D 结构（包含一个浮点 x，y 点）指定渐变的端点。如果用户希望当渐变到达终点时循环回起点颜色，那么用户应当将 cyclic 参数设置为 true。

当用户创建完 GradientPaint 对象以后，用户可以调用 Graphics2D 中的 setPaint 方法来使用它：

```
GradientPaint gradient = new GradientPaint(0, 0, Color.red, 50, 50, Color.blue, true);
  newGraphics.fill(rect);
```

TexturePaint 对象允许用户使用一幅图像来填充图形。用户只要提供一个包含用户想要用作填充图案的 BufferedImage 对象既可。正如用户将在本章后面看到的那样，BufferedImage 类是 Image 类的特殊版本，它使用数组来操作图像的单个像素。对于图像，用户需要提供一个描述填充图案尺寸的矩形和指示 TexturePaint 应该如何产生颜色的标志。

TexturePaint 对象只有一个构造器：

```
public TexturePaint(BufferedImage pattern, Rectangle2D rect2d, int interpolation)
```

pattern 参数指明该图像被用作填充图案。rect2d 参数描述了该图案的大小。换句话说，图案是按照由 rect2d 指定的尺寸复制的。interpolation 参数用来确定当填充不能表示图像中所有颜色时如何显示颜色。interpolation 的两个可能的值是 TexturePaint.BILINEAR 和 TexturePaint.NEAREST _ NEIGHBOR。程序清单 26.1 显示了一个小应用程序，这个小应用程序通过使用非常宽的笔划和填充纹理绘制一个简单的路径。它有些奇特，但是在本例中使用的填充纹理是一幅实际的图片。使用简单的位图图案填充是非常普通的。该程序使用了一幅名为 katyface.gif 的图像。在运行程序时应确保具有该名字的图像文件和该小应用程序在同一个目录中。

```
import java.awt.*;
import java.applet.*;
import java.awt.geom.*;
import java.awt.image.BufferedImage;
import java.net.URL;

public class TextureDemo extends Applet
{

    public void paint(Graphics g)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }

// Get the Graphics2D object

        Graphics2D newG = (Graphics2D) g;

// Create a path to draw

        GeneralPath path = new GeneralPath();

        path.moveTo(60, 0);
        path.lineTo(50, 300);
        path.curveTo(160, 230, 270, 140, 400, 100);

// Load an image to use as the texture

        URL imgURL = null;
        try {
            imgURL = new URL(getDocumentBase(),"katyface.gif");
        } catch (Exception ignore) {
        }

        Image img = getImage(imgURL);

        MediaTracker tracker = new MediaTracker(this);
        try {
            tracker.addImage(img, 0);
            tracker.waitForAll();
        } catch (Exception e) {
            e.printStackTrace();
        }

// Normally you would create a buffered image and set the individual
// pixels yourself, but you can also use an existing image. Rather than
// trying to convert the loaded image into a BufferedImage, it is easier
// (from a programming standpoint) to just get a Graphics2D object for
// the BufferedImage and draw the texture image into it
        BufferedImage buff = new BufferedImage(img.getWidth(this),
            img.getHeight(this), BufferedImage.TYPE_INT_RGB);
        Graphics tempGr = buff.createGraphics();

        tempGr.drawImage(img, 0, 0, this);

// The TexturePaint requires a rectangle defining the area to be filled
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// In this case, just use the image size.

        Rectangle2D rect = new Rectangle2D.Float(
            0, 0, img.getHeight(this),
            img.getWidth(this));

// Create the textured paint

        TexturePaint painter = new TexturePaint(buff, rect,
            TexturePaint.NEAREST_NEIGHBOR);

        newG.setPaint(painter);

// Create a VERY wide stroke (100 pixels) round off the corners and
// make the ends square
        BasicStroke stroke = new BasicStroke(100,
            BasicStroke.CAP_SQUARE, BasicStroke.JOIN_ROUND);
        newG.setStroke(stroke);

// Draw the original path
        newG.draw(path);
    }
}
```

图 26.8 显示了 TextureDemo 小应用程序的输出。

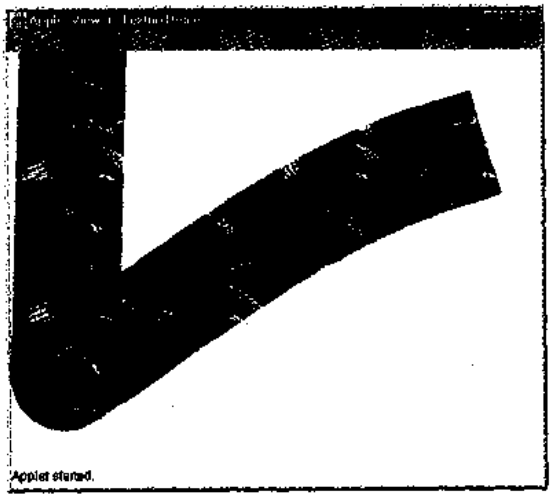


图 26.8 用户可以将一幅图像作为填充图案

26.6 转换

除了绘制形状以外，用户还经常需要移动图形、旋转图形和重新设置图形尺寸。这些操作

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

需要使用名为 AffineTransform 的对象来执行。AffineTransform 类包含一个矩阵，该矩阵用于将某个 x，y 点转换为另一个 x，y 点。如果用户确实喜欢使用自己的矩阵操作，那么用户可以通过提供各个用于转换的矩阵项来创建一个 AffineTransform。转换矩阵有三个列和两个行。如果用户将第 0 行元素编号为 m00、m01 和 m02，将第 1 行元素编号 m10、m11、和 M12，那么将 xorig，yorig 转换成 xnew，ynew 的公式将是：

```
xnew = xorig * m00 + yorig * m01 + m02
ynew = xorig * m10 + yorig * m11 + m12
```

幸运的是，对于不喜欢做矩阵运算的人可以直接使用 AffineTransform 类提供的方法建立转换。用户可以通过调用缺省的构造器创建一个空的 AffineTransform：

```
AffineTransform myTransform = new AffineTransform( )
```

用户可以通过调用 translate 方法增加一个转换(即位置移动)。下面的移动示例向左移动 5 个单位向下移动 10 个单位：

```
myTransform.translate( -5.0,10,0)
```

提示：当用户执行移动、旋转和缩放操作时，用户不必实际地执行转换。用户正在改变的是转换对点做的事情。将这些操作看成是移动转换是有一定帮助作用的。因为用户对转换所做的事情正是转换对形状所做的事情。

rotate 方法沿顺时针方向旋转该形状(当角度为负值时，沿逆时针方向旋转)。旋转的角度是按弧度给出的。用户可以选择地指定 x，y 点，当旋转转换不是绕着原点进行时则需要指定该旋转点。rotate 方法有下列几种形式：

```
public void rotate(double numRadians)
public void rotate(double numRadians,double x,double y)
```

scale 方法使用特定的值乘以 x 和 y 坐标。当用户调用 scale 方法时，用户总会为 x 和 y 传递一个缩放值。如果用户只希望沿一个方向缩放图像，那么就可在不希望缩放的方向使用 1.0 作为缩放值。要想只将 x 方向的转换尺寸放大一倍但不改变 y 方向的尺寸，那么就可以使用 2.0 作为 x 方向的缩放值，用 1.0 作为 y 方向的缩放值：

```
myTrauaform,scale(2.0,1.0)
```

Shear 方法类似于 scale 方法，只是它基于 y 值改变 x 值，或者基于 x 值改变 y 值。如果用户使用系数 2 剪切 x 值，那么每个点的 x 值都将增大剪切系数乘以该点的 y 值这么大的量。假设用户在 x 方向上剪切 2，那么点 5，10 将被剪切成(5 + 2 * 10)，10 或简写为 25，10。y 值越高，那么 x 移动得就越多。在 y 方向上的剪切关系也是相同的。shear 方法是按照如下所示的方式定义的：

```
public void shear(double xshear,double yshear)
```

只操作 AffineTransform 对象是不会改变屏幕显示的。用户需要将转换应用到要绘制的形状上或整个 Graphics2D 对象上。为了操作某个形状，需要调用 CreateTransformedShape 方法，该方法返回应用了当前转换的新形状。下面的代码片段将一个矩形旋转 45 度：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Restange2D rect = new Rectangle 2D.Float(0, 0, 50, 20);
AffineTransforn transrorm = new AffineTransform( );
Transform.rotate(45.0 * 3.1415927/180.0);
Shape rotatedRect = transform.create TransformedShape (rect);
```

提示:用户可以通过乘以 3.1415927/180.0 或 0.0174532928 将角度值转换成弧度值。

用户也可以通过调用 Graphics2D 的 setTransform 方法将转换应用于整个 Graphics2D 对象。转换 Graphics 对象本身可保证 Graphics 对象在被显示之前完成转换。setTransform 方法是按照如下所示的形式设置的:

```
public void setTransform(AffineTransform transform)
```

26.7 绘制文本

绘制文本在图形操作中是大树一棵(绘制形状和图像是另外两棵大树)。二维应用编程接口在原来的文本绘制例程中增加了一些方便的扩展。其中最值得注意的是旋转文本的能力。Graphics2D 类实现了 drawString 方法的几个不同的版本。最简单的版本使用一个字符串和一组 x,y 坐标作为参数:

```
public void drawString(String textString,float x,float y)
```

这个 drawString 的简单版本类似于 Graphics 类的 drawString,所不同的是这里的坐标使用浮点值而不使用整数。其它的不同当然是在显示该字符串之前将按照转换矩阵所做的转换应用到该字符串上。这种方法允许用户旋转文本串甚至用有趣的填充图案绘制文本。

26.7.1 有风格设置的字符串

用户也可以绘制 StyledString 对象,该对象中包含有一个字符串和一个与该字符串关联的属性集(一般只是字体)。用户甚至可以将多个有风格设置的字符串连接成为一个包含不同字体的文本字符串。用户可能希望打印极为常见的"Hello World"信息,例如,用户可以对每个单词使用不同的字体。下面的代码片段就是做这件事情况的:

```
Font fntCurier = new Font("Courier", Font.PLAIN, 48);
StyledString ssHello = new StyledString("Hello", fntCourier);
Font fntHelvetica = new Font("helvetica", Font.BOLD, 48);
StyledString ssWorld = new StyledString("World!", fntHelvetica);
StyledString helloWlrld = ssHello.concat(ssWorld);
NewGraphics.drawString(helloWorld, 100, 100);
```

像正常的 Java String 类一样,StyledString 是一成不变的——用户不能改变它的内容。但是用户可以通过调用 concat 方法(将两个字符串合并在一起)来创建该字符串的新版本。或者使用 substring 方法(获得字符串的一部分)也可以创建该字符串的新版本。substring 方法与 String substring 方法是相等的:

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
public StyledString substring(int startIndex, int endIndex)
```

26.7.2　文本布局

虽然 StyledString 类是很有用的，但最有用的新文本特性是 TextLayout 对象。Graphics 类中原来的 drawString 方法最大的缺点之一是它只能绘制单行文本。用户经常需要绘制整个文本段落。直到目前为止，开发人员还不得不自己创建一个把段落分成多行文本，并多次调用 drawString 方法的例程。TextLayout 类能够把文本格式化成多行文本，甚至可以处理从右到左和从顶向下的段落风格(比如竖版中文资料)，当处理非英文文本时可能用到这种风格。用户甚至可以使用 TextLayout 方法格式化 StyledString，允许段落中有多种字体。

TextLayout 类有几个构造器：

```
public TextLayout(String string, Font font)
public TextLayout(Sring string, AttributeSet attributes)
public TextLayout(StyledString text)
public TextLayout(AttributedCharacterIterator text)
```

两个最常用的构造器是使用 String 参数和 Font 参数的构造器和使用 StyledString 参数的构造器。如果用户希望使用更复杂的字符，那以用户可以使用一个字符属性集，或通过重复符为每个字符定义属性。程序清单 26.2 中的程序在屏幕上显示了一个简单的段落：

```
import java.awt.*;
import java.applet.*;
import java.awt.geom.*;
import java.awt.font.*;

public class Paragraph extends Applet
{
    public void paint(Graphics g)
    {
// Get the Graphics2D object

        Graphics2D newG = (Graphics2D) g;

        String message = "Harold, on the other hand, refused " +
            " to eat the chalk. He reached into his lunchbox " +
            " and removed a small block of balsa wood, which " +
            " he proceeded to chew on gleefully. Harold always " +
            " referred to balsa wood as a \"light snack\"";

        Font fntRoman = new Font("timesroman", Font.PLAIN, 24);

        TextLayout layout = new TextLayout(message, fntRoman);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            newG.drawString(layout, (float) 100, (float) 100);
        }
    }
```

26.7.3 字符属性

虽然用户可以为能够改变字符的字体而感到高兴,但是对改变其它属性的要求会经常地出现。StyledString 类和 TextLayout 类都允许使用指定字符属性集,字符属性集定义的东西远不只字体一项。用户可以使用 AttributeSet 对象来指定字符属性。AttributeSet 对象和 StyledString 对象一样是永远不变的。如果用户要修改它,那以用户将如何设置它呢? 这里有一个 MutableAttributeSet 对象可以帮助用户做这种修改。用户可以典型地创建一个 MutableAttributeSet 对象,为它增加新的属性,然后将它传递给一个使用 AttributeSet 作为参数的方法。用户可以把 AttributeSet 和 MutableAttributeSet 之间的关系看作类似于 String 和 StringBuffer 之间的那种关系。使用 MutableAttributeSet 可以构造无法改变的 AttributeSet。

MutableAttributeSet 本身是 AttributeSet 的子类,用户通常将它传递给期望 AttributeSet 对象的方法。如果某个方法需要坚持引用 AttributeSet,那么该方法将产生一个 AttributeSet 的拷贝。否则,因为该集是实际可变的集,所以用户可以在以后的程序中修改它,这种修改将会影响到已有的属性。因此,AttributeSet 只有当在该类中没有修改它的方法时才是不可改变的。如果该类实际地拥有 MutableAttributeSet,那么 AttributeSet 是可以改变的。

为了设置字符属性,用户需要使用 TextAttributeSet 对象。TextAttributeSet 对象是一个可变的对象集,它定义了许多常量。某些常量代表了某些文本属性的名字,而其它常量描述了那些属性缺省的或常用的值。

有效的文本属性名如下所示:

属 性 名	含 义
LANGUAGE	用于文本的语言(通常是一个地点)
READING	单词的发音信息(用于某些要求发音注释的语言)
INPUT_METHOD_SEGMENT	用于把一行分成几段(通常是分成单词)
SWAP_COLORS	是否交换文本的前景色和背景色
FAMILY	字体族
WEIGHT	字符的粗细(粗体文本意味着更粗)
POSTURE	文本的倾斜
SIZE	字体的点阵
TRANSFORM	应用于字体的图形转换
FONT	用于字符的 Font 实例
BIDI EMBEDDING	双向文本控制

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

续表

属　性　名	含　义
BACKGROUND	指示文本背景颜色的 Color 实例
FOREGROUND	指示文本前景颜色的 Color 实例
UNDERLINE	指示文本是否使用下划线
STRIKETHROUGH	指示文本是否使用删去线
SUPERSUBSCRIPT	指示文本是上标或下标
JUSTIFICATION	调整用在对齐中的空白量
RUN_DIRECTION	对文本是否从左到右、从右到左或从顶到底运行的控制
BIDI_NUMERIC	对罗马数字布局双向控制
BASELINT	调整所有字符的基线

为了设置不同的属性，用户只要使用属性名和它的值调用 add 方法即可。许多属性值拥有预设的常量；其它属性使用字符串或数字参数。下面的代码片段使用 Times-Roman 字体和删除线创建了一个文本属性集：

```
Fout fntRoman = new Font("timesroman",Font.PLAN, 24);
TextAttributeSet textAttr = new TextAttributeSet( );
TextAttr.add(TextAttributeSet.FONT, fntRoman);
TextAttr.add(TextAttributeSet.STRIKETHROUGH,
TextAttributeSet.STRIKETHROUGH_ON);
```

26.8　绘制图像

Java 二维应用编程接口清楚地认识到操作图像需要对图像执行各种数学运算。这种数学运算可以增强逐像素地处理图像数据的能力。虽然 AWT 的早期版本并不阻止这种操作，但它们不是很有效，并且许多操作需要人工执行。

26.8.1　缓冲区中的图像

在某些版本的 Java 中，逐像素地操作图像是非常困难的。用户必须或者创建图像过滤器并用图像过滤器修改像素，或者创建像素抓取器抓取图像后创建一个 MemoryImageSource 并将像素数组转换成一幅图像。BufferedImage 类通过提供可以直接操作像素的图像来提供快速方便的捷径。

创建缓冲区中的图像最容易的办法是指明宽度，高度和像素类型：

```
public BufferedImage(int width, int height, int pixelType)
```

像素类型有许多不同的选项，但是用户通常只需要 TYPE_INT_ARGB 或 TYPE_INT_RGB。TYPE_INT_RGB 像素类型与在 Java 早期版本中使用的像素类型是一样的。每个

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

像素使用一个 32 位整数描述,前八位是透明度,接下来八位是红色分量,再接下来八位是绿色分量,最后八位是蓝色分量,排放顺序是 aarrggbb。TYPE _ INT _ RGB 格式除了没有透明度外几乎与 TYPE _ INT _ ARGB 一样。相当于 aa 部分始终是 255。如果用户工作的系统使用 bbggrr 格式的整数编码颜色,那么用户可以对像素使用 TYPE _ INT _ BGR 格式。给出的三个整数 red、green 和 blue 的值,用户可以按 RGB 格式将他们编码如下:

```
int rgb = (red< <16) + (green< <8) + blue;
```

为了使用 bgr 格式存储像素,只要颠倒一下变量顺序即可,例如:

```
int bgr = (blue< <16) + (green< <8) + red;
```

要想从 ARGB 像素中抽取阿尔法分量、红色分量、绿色分量和蓝色分量,用户可以使用下列语句:

```
int alpha = (argb> >24)&225;
int red = (argb> >16)&225;
int green = (argb> >8)&225;
int blue = argb&225;
```

BufferedImage 的 getRGB 方法不管像素在图像中如何存储都返回一个 ARGB 格式的像素:

```
public int getRGB(int x,int y)
```

要想在缓冲区图像中设置一个像素,用户可以调用 setRGB 方法(不要忘记使用像素的 ARGB 格式):

```
public int setRGB(int x,int y,int rgb)
```

26.8.2 将图像拷贝到 BufferedImage 中

当用户使用 java.applet.Applet 或 java.awt.Toolkit 中的 getImage 方法时,用户取回的图像仅仅是一个 Image 对象。可以有许多将图像转换成 BufferedImage 的方法,但大部分方法都很复杂。一种较快速的方法虽然不是非常高效的,但却是非常容易的,并且如果用户只需要做一次转换,那么效率也不会成为严重的问题。

下面的代码片段创建了一个 BufferedImage 对象并获得一个绘制缓冲区图像的 Graphics 对象,然后再将已有的一幅图像绘制到缓冲区图像上:

```
//用已有图像的宽度和高度创建缓冲区图像
BufferedImage buff = new BufferedImage(img.getwidth (this),
img.getHeight(this), BufferedImage.TYPE _ INT _ RGB;
//获得一个绘制缓冲区图像的 Graphics 对象
Graphics tempGr = buff.createGraphics();
//将已有的图像绘制到缓冲区图像上
tempGr.drawImage(img,0,0,this);
```

26.8.3 过滤缓冲区里的图像

图像过滤器是在 Java1.0 中引入的,到目前已经成为操作图像的有效手段。Java 二维应

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间:可以联系群主获取更多大型企业内部技术教程。

用编程接口使用了几个预制的过滤器扩展了已有的图像过滤功能，并且具有了对缓冲区图像过滤的能力。老的 AWT 图像过滤机制使用了生产者-消费者模型，其中过滤器既是源图像的消费者又是过滤后的图像的生产者。虽然 BufferedImage 类包含一个 getSource 方法，当创建 FilteredImageSource 时要求 getSource 方法，该方法的返回值为 null。但是，用户不能实际地使用已有的生产者-消费者接口过滤缓冲区里的图像。取而代之的 BufferedImageOP 接口定义了一个方法可以通过直接操作缓冲区里的像素将整个缓冲区图像源过滤成图像目标。

BufferedImageFilter 类是作为将图像操作与已有的生产者-消费者模型挂接起来的手段而存在的。

BufferedImageOP 仅仅是一个接口，所以用户不能直接对它实例化。用户必须创建一个实现该接口的类，或者使用一个已有的图像处理类：AffineTransformOp、BandCombineOp、ColorConvertOp、ConvolveOp、RescaleOp 和 Thresholdop。某些内置的操作允许源和目标一致。其它操作要求分别的图像。比较粗糙的方法是，如果在转换操作中使用了不只一个像素，或者如果像素从某个位置移动另一个位置，那么用户就需要把源和目标分开。例如，颜色转换几乎从不需要分成源图像和目标图像。

要想使用某个 BufferdImageOp 类过滤某个图像，用户只要创建处理方式并使用源图像和目标图像调用 filter 方法即可（某些处理方式可以使用同一幅图像）。下面是调用 filter 方法的示例：

```
Op.filter(srcBuff,destBuff);
```

如果用户只需要绘制一幅过滤完的图像，那么 Graphics2D 里的 drawImage 方法允许用户绘制由 BufferedImageOp 过滤的图像：

```
public void drawImage(BufferedImage image,BufferedImageOp op,ImageObserver obs)
```

AffineTransformOp AffineTransformOp 对图像执行映射转换。这种转换可以包括缩放、旋转和移动的任意组合。下面的片段创建一个 AffineTransformOp 对象，它将图像旋转 45 度，例如：

```
AffineTransform transform = new AffineTransform();
transform rotate(45.0 * 3.1415927/180.0,buff.getWidth()/2,buff.getHeight()/2);
AffineTransformOp op = new BilinearAffineTransformOp(transform);
```

> **注意**：AffineTransformOp 是一个抽象类。用户必须选择 BilinearAffineTransformOp 和 NearestNeighborAffineTransformOp 中的一个。这两个版本的差异只在于执行转换时选择颜色的方式。

AffineTransformOp 要求独立的源图像和目标图像。

BandCombineOp 乍一看 BandCombineOp 过滤器可能有些陌生并在感觉上认为它没有什么用。缓冲区图像是被存放在光栅对象中的，这种图像实际上只是像素数组。在这些光栅中是一些颜色“频带”。对于典型的 RGB 图像，有三个频带：红色、绿色和蓝色。BandCombineOp 过滤器允许用户通过各种颜色频带的结合来改变图像的颜色。对于三频带图像，用户可以指定一个三行四列矩阵，例如：

```
B11 B12 B13 B1OFFSET
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

B21 B22 B23 B2OFFSET
B31 B32 B33 B3OFFSET

计算频带 1 的颜色的公式是：

```
B1COLOR = B1COLOR * B11 + B2COLOR * B12 + B3COLOR * B13 + B1OFFSET
```

下面的过滤器对图像不做任何改动(这是一个频带结合的恒等矩阵)，因为它对频带中的每个颜色都乘以 1。

```
1.0    0.0    0.0    0.0
0.0    1.0    0.0    0.0
0.0    0.0    1.0    0.0
```

在典型的 RGB 光栅中，频带 1 是红色，频带 2 是绿色，频带 3 是蓝色。下面的矩阵从图像中删除绿色，只保留红色和蓝色：

```
1.0    0.0    0.0    0.0
0.0    0.0    0.0    0.0
0.0    0.0    1.0    0.0
```

用户也可以反转颜色。为了反转红色，矩阵中的红色行应该是：

```
-1.0    0.0    0.0    255.0
```

这一行矩阵项将使图像中的红色乘以 -1 再加 255 后产生新的红色值，用公式可以表示为 255 - red。

真正令人感兴趣的结合是让一种颜色在另一种颜色中起作用。例如，用户可以过滤红色使得它成为图像中红色、绿色和蓝色分量的组合。下面的矩阵不改变红色和蓝色，但是使绿色由 1/2 的绿色分量、1/4 的红色分量和 1/4 的蓝色分量组合而成。换句话说，红色和蓝色越亮则绿色也越亮。这里是该矩阵：

```
1.0     0.0     0.0     0.0
0.25    0.5     0.25    0.0
0.0     0.0     1.0     0.0
```

下面的代码段创建了一个 BANDCOMBINEOP 对象：

```
float filt[][] = {
{1.0f,    0.0f,    0.0f,    0.0f}
{0.25f,   0.5f,    0.25f,   0.0f}
{0.0f,    0.0f,    1.0f,    0.0f}};
BandCombineOp op = new BandCombineOp(filt);
```

BandCombineOp 可以使用同一个图像作为源和目标。

ColorConvertOp ColorConvertOp 类将图像由一种颜色空间转换为另一种颜色空间。颜色空间定义了颜色如何表示。用户可能对 RGB 颜色空间比较熟悉，在 RGB 颜色空间中，用户可以使用颜色中的红色分量、绿色分量和蓝色分量来指定颜色。出版界经常使用的颜色空间叫做 CMYK，它使用青色、品红色、黄色和黑色分量指定颜色。下面的代码片段创建一个将图像转换成灰度图像的 ColorConvertOp：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
ColorSpace graySpace = ColorSpace.getInstance(ColorSpace.CS_GRAY);
ColorConvertOp op = new ColorConverop(graySpace);
```

ConvolveOp　ConvolveOp 类实现了一种普通的图像操作，它基于周围像素使用某个简单矩阵运算修改每个像素。当用户创建一个 ConvolveOp 对象时，用户需要提供一个 Kernel 对象，该对象中包含将要应用到图像上的矩阵。Kernel 对象甚至包含有几个预定义的使用边缘检测来提高图像质量的尖锐化矩阵。

如果用户使用过拥有不同的图像算法的绘画程序，那么用户可能看到过一种叫做"边缘检测"的功能。当用户在图像上运行边缘检测时，用户会得到一幅含有显示图像中某些边缘的转黑图像。用户可以使用 Kernel 和 ConvoleOp 创建一个边缘检测。

当用户为图像处理创建一个 Kernel 时，用户实际上只要指定一个用于计算图像中每个像素的颜色的矩阵就可以。该矩阵决定周围像素如何影响当前像素。在边缘检测中，用户完全忽略了当前像素的颜色。实际上用户使用的是左上角像素和右下角像素（用户也可以使用其它相对的像素）。把左上角像素乘以一个系数，而把右下角像素乘以负的该系数。系数越大，那么边缘就越明显。

Kernel 对象的矩阵应该有奇数的阶数，并且通常为 3×3 阶矩阵。矩阵中心的值是当前像素的乘法因子。周围的值是周围像素的乘法因子。为了执行边缘检测，用户必须忽略当前像素，这样中心值应该为 0。矩阵的左上角和右下角包括边缘检测的乘法因子。图 26.9 显示了一幅图像和该图像的边缘检测。

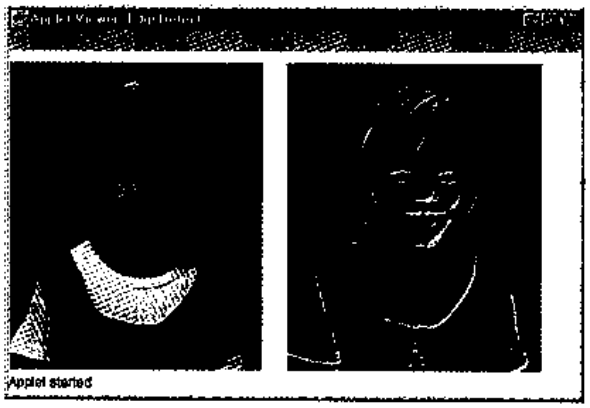


图 26.9　边缘检测显示了颜色之间的尖锐化过渡

下面的代码片段使用乘法因子 5 创建了一个更强的边缘检测器：

```
float matrix[] = {
-5,      0,      0,
 0,      0,      0,
 0,      0,      5,};
kernel kernel = new kernel(3,3, matrix);
ConvolveOp op = new ConvolveOp(kernel,ConvolveOp.EDGE_ZERO_FILL);
```

Kernel 类有几个预定义的图像尖锐化矩阵。这些矩阵是 SHARPEN3x3_1、SHARPEN3x3_2、SHARPEN3x3_3。这些矩阵定义如下：

```
SHARPEN3x3_1:
-1       -1       -1
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
 -1      9     -1
 -1     -1     -1
SHARPEN3x3_2:
  1     -2      1
 -2      5     -2
  1     -2      1
SHARPEN3x3_3:
  0     -1      0
 -1      5     -1
  0     -1      0
```

用户可以使用这些预定义矩阵之一创建一个矩阵，例如：

```
Kernel sharpen = new Kernel(Kernel.SHARPEN3x3_1);
```

当用户创建完 Kernel 之后，用户就可以使用它来创建 ConvolveOp：

```
ConvolveOp op = new ConvolveOp(sharpen);
```

因为它依赖于周围像素，所以 ConvolveOp 必须有不同的源图像和目标图像。

LookupOp LookupOp 类提供了一个简单的检查表，该表将一个像素值映射成为另一个像素值。为了创建一个 LookupOp，用户需要创建一个 LookupTable，它有一个数组或字节或短型值。Lookup Table 本身是一个抽象类。用户必须创建 ByteLookupTable 和 ShortLookupTable 两者中的一个。下面的代码片段创建了一个 LookupOp，它的作用是反转颜色(0 变成 255，1 变成 254，等等)

```
short lookupValues[] = new short[256];
for (int i = 0; i<lookupValue.length;i++){
    lookupValues[i] = (short)(255-i);
}
LookupTabl table = new ShortLookupTable(0,lookupValues);
LookupOp op = new LookupOp(table);
```

图 26.10 显示了这种颜色翻转前后的图像效果。



图 26.10 检查表可以快速地反转颜色

RescaleOp RescaleOp 类允许用户基于一个系数和一个偏移量修改图像的颜色。该类的名字可能引导用户认为该类改变了图像的尺寸，但其实不然(可以使用 AffineTransformOp 改变图像尺寸)。用户可以通过提供缩放系数和偏移量来创建 Rescaleop。图像中每个颜色都乘以缩放系数并加上一个偏移量。Rescaleop 的构造器声明如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public RescaleOp(float factor,float offset)
public RescaleOp(float[] factor, float[] offsets)
```

当用户传递一个值数组给构造器时,这些值将被应用到每个光栅通道上。(对于 RGB 图像,光栅通道指的是红色、绿色和蓝色通道。)如果用户传递单个值,那么它们将应用于所有的通道。用户可以通过将颜色值减半后加 128 来产生水洗效果。下面的代码片段创建了一种水洗效果的 Rescaleop:

```
RescaleOp op = new RescaleOp(0.5 f,128);
```

图 26.11 显示了同一幅图像水洗处理前后的效果。



图 26.11　RescaleOp 可以产生水洗效果

因为它只操作单个像素,所以 RescaleOp 可以使用同一幅图像作为源和目标。

警告:RescaleOp 不调整大于 255 和小于 0 的颜色值。它把这些值原封不动地保留成最后的 RGB 颜色值,这将产生不可思议的畸变效果。如果用户使用 1 或小于 1 的系数(应该大于 0,译者注),那么就不会出现任何问题。应力图避免颜色值小于 0 和大于 255。

ThresholdOp　ThresholdOp 提供了一个开关型的颜色过滤器。用户可以指定一个临界值、一个低值和一个高值。任何小于临界值的颜色被指定为低值,所有大于临界值的颜色被指定为高值。这种指定是基于各通道进行的。(即在 RGB 图像中是按红色、绿色和蓝色通道处理的)用户可以通过提供由临界值、低值和高值组成的数组来为各颜色通道指定不同的值。正如用户可能猜测到的那样,ThresholdOp 实际上减少了图像中的颜色数。ThresholdOp 的构造器如下所示:

```
public ThresholdOp(float threshold,float low,float high)
public GhresholdOp(float[]thresholds,float[] lows, float[] highs)
```

图 26.12 显示了一幅临界值为 128、低值为 0 和高值为 255 的图像。用户可能认为将会产生一幅纯粹的黑白图片,但是其中还有一些颜色碎片。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。



图 26.12 ThresholdOp 严重地减少了图像中的颜色种类

26.8.4 操作缓冲区图像

有时用户可能希望不通过过滤器操作缓冲区图像里的像素。这时用户可以使用 getRGB 方法和 setRGB 方法直接操作像素。程序清单 26.3 显示了一个小应用程序，该小应用程序对图像执行了浮雕效果操作。该操作类似于边缘检测，但是不能使用 ConvolveOp 实现，因为它需要在对左上角和右下角乘以系数之前将当前像素设置为 128。应确保 Katyface.gif 文件与该小应用程序在同一个目录中。

```
import java.awt.*;
import java.applet.*;
import java.awt.geom.*;
import java.awt.font.*;
import java.awt.inage.*;
import java.net.URL.;

public class Emboss extends Applet
{

    public void paint(Graphics g)
    {

// Get the Graphics2D object

        Graphics2D newG = (Graphics2D) g;

// Load an image to display

        URL imgURL = null;
        try {
            imgURL = new URL(getDocumentBase(), "katyface.gif");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        } catch (Exception ignore) {
        }

        Image img = getImage(imgURL);

        MediaTracker tracker = new MediaTracker(this);
        try {
            tracker.addImage(img, 0);
            tracker.waitForAll();
        } catch (Exception e) {
            e.printStackTrace();
        }

// Normally you would create a buffered image and set the individual
// pixels yourself, but you can also use an existing image. Rather than
// trying to convert the loaded image into a BufferedImage, it is easier
// (from a programming standpoint) to just get a Graphics2D object for
// the BufferedImage and draw the image into it

        int width = img.getWidth(this);
        int height = img.getHeight(this);

// Create a buffered version of the image by creating a graphics
// context and drawing into it.

        BufferedImage buff = new BufferedImage(width,
            height, BufferedImage.TYPE_INT_ARGB);

        Graphics tempGr = buff.createGraphics();

        tempGr.drawImage(img, 0, 0, this);

// Create a buffered image to hold the resulting embossed image

        BufferedImage outBuff = new BufferedImage(width,
            height, BufferedImage.TYPE_INT_ARGB);

        embossImage(buff, outBuff);

        newG.drawImage(outBuff, 100, 100, this);
    }

// To emboss an image, you start with a completely gray destination image.
// For each pixel in the source image, look at pixels to the upper-left and
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// lower-right. Figure out the change in red, green, and blue between the
// upper-left and lower-right and look at the maximum change (either maximum
// positive or maximum negative) for any color component. For example,
// if the green changed by -5, blue changed by 10 and red changed by
// -100, the maximum change would be -100 (the red, which changed the most).
//
// Now, add the amount of change to 128 (the gray level) and create a
// pixel in the destination image with red, green, and blue values equal
// to the new gray level. Make sure you adjust the gray level so it can't
// be less than 0 or more than 255.
//
//
    public void embossImage(BufferedImage srcImage, BufferedImage destImage)
    {

        int width = srcImage.getWidth();
        int height = srcImage.getHeight();

// Loop through every pixel

        for (int i=0; i < height; i++) {
            for (int j=0; j < width; j++) {

// Assume that the upper-left and lower-right are 0
                int upperLeft = 0;
                int lowerRight = 0;

// If the pixel isn't on the upper or left edge, get the upper-left
// pixel (otherwise, the upper-left for edge pixels is the default of 0)

                if ((i > 0) && (j > 0)) {

// The & 0xffffff strips off the upper 8 bits, which is the transparency

                    upperLeft = srcImage.getRGB(j-1, i-1)
                        & 0xffffff;
                }

// If the pixel isn't on the bottom or right edge, get the lower-right
// pixel (otherwise, the lower-right for egde pixels is the default of 0)

                if ((i < height-1) && (j < width-1)) {

// The & 0xffffff strips off the upper 8 bits, which is the transparency
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                lowerRight = srcImage.getRGB(j + 1, i + 1)
                    & 0xffffff;
            }

// Get the differences between the red, green and blue pixels
            int redDiff = ((lowerRight > > 16) & 255) -
                ((upperLeft > > 16) & 255);
            int greenDiff = ((lowerRight > > 8) & 255) -
                ((upperLeft > > 8) & 255);
            int blueDiff = (lowerRight & 255) -
                (upperLeft & 255);

// Figure out which color had the greatest change

            int diff = redDiff;
            if (Math.abs(greenDiff) > Math.abs(diff))
                diff = greenDiff;
            if (Math.abs(blueDiff) > Math.abs(diff))
                diff = blueDiff;


// Add the greatest change to a medium gray
            int greyColor = 128 + diff;

// If the gray is too high or too low, make it fit in the 0-255 range
            if (greyColor > 255) greyColor = 255;
            if (greyColor < 0) greyColor = 0;

// Create the new color, and don't forget to add in a transparency
// of 0xff000000 making the image completely opaque

            int newColor = 0xff000000 + (greyColor < < 16) +
                (greyColor < < 8) + greyColor;

            destImage.setRGB(j, i, newColor);
        }
    }
}
```

图 26.13 显示了一幅经过浮雕化处理的图像。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

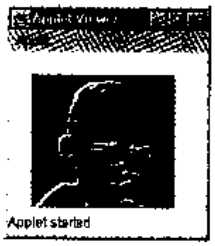


图 26.13 用户可以使用 BufferedImage 创建浮雕效果

26.9 透明

在 Java 的早期版本中，AWT 的图像应用编程接口允许使用透明和半透明像素。一种颜色中除了规则的红绿蓝值以外还有第四个值—阿尔法分量。阿尔法分量为 0 意味着该颜色是完全透明的(即，该 RGB 值实际未被使用)。阿尔法分量为 255 意味着该颜色是完全不透明的。当该颜色被画在一个已有的像素上时，该像素将被完全地隐藏起来。0 到 255 之间的阿尔法值将会产生十分有趣的结果。用户可以将两幅图像互相覆盖并混合在一起。

Java 的二维图形应用编程接口将透明模式扩展到所有绘画上，它允许用户为任何颜色指定阿尔法分量。

用户将在 Java 文档中经常遇到的一个术语“premultiplied alpha”，即预乘阿尔法。当 Java 绘制包含阿尔法分量的像素时，实际上它在为画在某个特定位置上的每个不同的像素增加各种红、绿和蓝值。当增加这些值时，Java 首先用它们各自的阿尔法值乘它们。如果图像是完全透明的，它的阿尔法值为 0。那么在代码中就没有必要做处理，因为阿尔法为 0 时不必绘制任何像素。如果要实际处理，那么就把红、绿和蓝色分量分别乘以阿尔法值 0 后再加到当前像素值上。因为所有乘法运算的结果都是 0，所以它们对当前颜色根本不起作用。

有时，一种颜色拥有“预乘阿尔法”意味着红、绿和蓝色分量已经被阿尔法分量乘过。它们已经做好了加进当前像素颜色的准备。用户不必每次都预乘阿尔法值，但是除非用户明确地希望如此。缺省的颜色处理将分别存放红、绿和蓝色分量。

用户可以使用下列构造器创建带透明分量的颜色：

```
public Color(int red, int green, int blue, int alpha)
public Color(int rgba, boolean hasAlpha)
public Color(float red, float green, float blue, float alpha)
```

程序清单 26.4 里的程序绘制了两个图形，其中一个使用了部分透明的红色，另一个使用部分透明的蓝色。它们产生洋红色的覆盖区域。如果没有阿尔法分量，就不会产生覆盖，并且第一个三角形将在第二个三角形被画出时隐藏起来。

```
import java.awt.*;
import java.applet.*;
import java.awt.geom.*;
import java.awt.image.BufferedImage;
import java.net.URL;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public class AlphaDraw extends Applet
{

    public void paint(Graphics g)
    {
        Graphics2D newG = (Graphics2D) g;

// Create a partially transparent blue

        Color transBlue = new Color(0, 0, 255, 128);
        newG.setColor(transBlue);

        GeneralPath path = new GeneralPath();

        path.moveTo(60, 0);
        path.lineTo(50, 300);
        path.curveTo(160, 230, 270, 140, 400, 100);

        newG.fill(path);

        Color transRed = new Color(255, 0, 0, 128);
        newG.setColor(transRed);

        path = new GeneralPath();
        path.moveTo(200, 300);
        path.lineTo(10, 100);
        path.lineTo(300, 40);
        path.lineTo(200, 300);

        newG.fill(path);

    }
}
```

26.10　裁剪

在 Java 的二维应用编程接口中，用户现在可以使用任意 Shape 对象设置裁剪区，包括文本对象。在本章的前面用户看到了如何设置图像的 paint 对象，当用户绘制文本时 paint 方法使用图像来填充文本。用户可以通过创建由文本或用户喜欢的形状组成的模板来做许多类似的事情。需要用户做的工作是创建形状和调用 Graphics2D 中的 clip 方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 26.5 显示了如何使用文本字符作为裁剪区。注意在裁剪之前用户需要移动它。要确保 johnpat2.gif 文件与用户的小应用程序在同一个目录里。

```
import java.awt.*;
import java.applet.*;
import java.awt.geom.*;
import java.awt.font.*;
import java.awt.image.BufferedImage;
import java.net.URL;

public class TextClip extends Applet
{

    public void paint(Graphics g)
    {

// Get the Graphics2D object

        Graphics2D newG = (Graphics2D) g;

// Load an image to use as the texture

        URL imgURL = null;
        try {
            imgURL = new URL(getDocumentBase(), "johnpat2.gif");
        } catch (Exception ignore) {
        }

        Image img = getImage(imgURL);

        MediaTracker tracker = new MediaTracker(this);
        try {
            tracker.addImage(img, 0);
            tracker.waitForAll();
        } catch (Exception e) {
            e.printStackTrace();

    }

        int width = img.getWidth(this);
        int height = img.getHeight(this);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        Font bigfont = new Font("Serif", Font.BOLD, 60);

        StyledString ssGrahams = new StyledString("The Grahams",
            bigfont);

// Set the clipping area to be the text string
        AffineTransform transform = new AffineTransform();
        transform.translate(0, 100);

        Shape clipShape = transform.createTransformedShape(
            ssGrahams.getStringOutline());

        newG.clip(clipShape);

// Draw the image over the clipped area
        newG.drawImage(img, 0, 0, width * 2, height * 2, this);

    }
}
```

Java 的二维应用编程接口提供了非常丰富的图形功能集，可以大大地减轻程序员的负担。使用这个新的二维应用编程接口，用户可以创建极其漂亮的直观显示，可以更好地显示信息，并且可以大大地提高绘图效率。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第四部分　输 入 输 出

第 27 章　文件和流
第 28 章　使用字符串和文本
第 29 章　使用多国语言实现
第 30 章　通信和网络
第 31 章　TCP 套接字
第 32 章　UDP 套接字
第 33 章　java.net
第 34 章　java 安全性深析
第 35 章　对象序列化
第 36 章　远程方法调用
第 37 章　管理 API

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第27章 文件和流

本章内容：

27.1 什么是流
27.2 基本输入和输出类
27.3 处理文件
27.4 使用管道
27.5 缓冲区里的流
27.6 数据流
27.7 字节数组流
27.8 字符数组流
27.9 字节和字符的转换
27.10 StringBufferInputStream
27.11 对象流
27.12 其它流
27.13 StreamTokenizer类

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

27.1 什么是流

所有的计算机程序必须接受输入和产生输出。很明显，每个计算机语言必须拥有处理输入和输出的方式，否则它就无法编写有用的程序。丰富的类集是 Java 的特征，Java 的类可以描述一切输入和输出，从普通的输入流和输出流到复杂的随机访问文件。现在用户可以有机会体验这些重要的类。

java.io 包为数据读取和写入提供了不同的输入和输出流。有些流是用于各种资源的，包括对普通文件系统功能的访问(比如文件和目录的创建、删除和重命名以及目录列表)。输入和输出流可以被连接到文件、网络套接字或内部的内存缓冲区。

java.io 包还包含了许多流过滤器。流过滤器允许用户以各种不同格式访问数据流。用户也可以创建自己的过滤器以增加额外的功能。

计算机系统使用的所有信息都从输入经过计算机流向输出。这种数据流动的概念引出了术语“Stream”(即流)。流就是数据流动。输入流指引来自外部世界的数据从键盘或文件流向计算机；输出流指引数据流向输出设备，比如屏幕和文件。因为流实际上是笼统的，基本的流并不特别地定义数据流入和流出的设备。就和传送驱动灯泡、电视机或洗碟机的电流的电线一样，基本输入和输出流可以被指向或来自许多不同的设备。

在 Java 中，流是用类来描述的。这种类有许多，其中最简单的类描述了基本输入和输出流(InputStream and OutputStream)，基本输入输出流提供了一般的流动能力。Java 从这些基本类派生出其它面向特定类型的输入和输出的类。用户可以在 java.io 包中找到所有这些类：

- Input Stream　　基本输入流。
- BufferedInputStream　　基本的缓冲区输入流。
- DataInputStream　　读取原始数据类型的输入流。
- FileInputStream　　用作基本文件输入的输入流。
- ByteArrayInputStream　　源为字节数组的输入流。
- StringBufferInputStream　　源为字符串的输入流。
- LineNumberInputStream　　支持行号的输入流。
- PushbackInputStream　　允许在字节被读取之后将该字节压回流中的输入流。
- PipedInputStream　　用作线程通信的输入流。
- SequenceInputStream　　组合其它两个输入流的输入流。
- OutputStream　　基本输出流。
- PrintStream　　显示文本的输出流。
- BufferedOutPutStream　　基本的缓冲区输出流。
- DataOutPutStream　　用作写入原始数据类型的输出流。
- FileOutputStream　　用作基本文件输出的输出流。
- FilterInputStream　　用于在已有输入流类中增加新行为的抽象输入流。
- FilterOutputStream　　用于在已有输出流类中增加新行为的抽象输出流。
- ByteArrayOutputStream　　目标为字节数组的输出流。
- PipedOutputStream　　用作线程间通信的输出流。
- File　　封装磁盘文件的类。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- FileDescriptor　　包含关于文件的信息的类。
- RandomAccessFile　　封装随机访问磁盘文件的类。
- StreamTokenizer　　允许用户将流作为一系列语言字符来输入。

很明显，本章要介绍的类实在是太多了。单就 Java 输入输出可以编写一本书。由于这个原因，本章着重介绍最有用途的流类，尤其是基本输入输出和文件处理以及线程间通信。本章从这些类的简介开始使用一些示例程序来演示这些类是如何工作的。

27.2　基本输入输出类

较为特殊的 Java 流类比如 FileInputStream 和 ByteArrayOutputStream 都依赖于普通的基本类 InputStream 和 OutputStream 的基本功能；如果进一步扩展它们将会创建出更加特殊的类。因为 InputStream 和 OutputStream 都是抽象类，所以用户不能直接使用它们。但是因为所有 Java 流类在它们的氏族树上都拥有 InputStream 或 OutputStream，所以用户应该知道这些类必须提供什么。

27.2.1　InputStream 类

InputStream 类描述了基本的输入流。为此，它定义了所有输入流需要的方法集。表 27.1 列出了这些方法，但没有带上它们的参数。

表 27.1　InputStream 类的方法

方　法	描　述
read( )	将数据读入流中
skip( )	在流中跳过若干字节
available( )	返回流中当上可用的字节数
mark( )	在流中标志位置
reset( )	返回在流中被标志过的位置
markSupported( )	返回表示该注解是否支持标志和复位的布尔值
close( )	关闭流

在该类中通过对 read( )方法重载提供了三种从流中读取数据的方法。这些方法的签名如下：

```
int read( )
int read(byte b[ ])
int read(byte b[ ],int off,int len)
```

从 InputStream 对象中获取数据的最基本方法是 read 方法。

```
public abstract int read( ) throws IOException
```

该方法从输入流读取单个字节并返回它。该方法执行阻塞读取，阻塞读取意味着如果没有可用数据，那么该方法就一直等待。所以当某个数据源没有要被读入的数据时，该方法在返回之前一直等待某个字节变成可用。举例来讲，就是当流位于网络上并且下一个数据字节尚

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

未到达。用户必须注意这种情况,否则如果不关心这种情况可能会引起类似于在第 13 章"线程"中讨论过的那种问题。

当流到达文件的结尾时该方法返回 -1。注意,为了能够返回数据的全部 8 位(一个字节)并且只在流到达结尾时仍然拥有 -1;这些值将被实际地返回,好像它们是从无符号字节中产生的,即如果用户在流中写入一个 -1,当读取它时将返回一个值为 255 的整数。幸运的是通过将流的 int 型值强制成 byte 型数据就能够得到 -1。

注意:这个 read 方法是最重要的方法,因为它是实际地从本地源中抓取数据的方法。所有的其它方法都使用这个方法执行它们的读取工作。

```
public int read(byte[ ] bytes) throws IOException
```

该方法用从流中读出的字节填充数组并返回读取字节的数量。读取的字节数比该数组能够容纳的字节数少是经常的,因为流中可能没有足够的字节来填充它。当流到达文件结尾时该方法返回 -1。在到达文件结尾之前,用户将一直接收流中的所有字节。换句话说,如果流中剩有 50 个字节,而用户请求读取 100 个字节,那么这个方法将返回 50 个字节,并在下一次调用它时返回 -1。

```
public int read(byte[ ] bytes, int offset, int length) throws IOException
```

该方法使用流中最多 length 个字节并从数组的 offset 位置开始填充数组。该方法正常情况下返回读取的字节数,但在文件结束处返回 -1。

当没有可用数据时 read 方法总是阻塞(一直等待而不返回)。为了避免阻塞,可能需要询问用户可以安全无阻塞地读取多少字节数。available 方法返回的就是这个字节数:

```
public int avaible( ) throws IOException
```

用户可以通过向 skip 方法传递用户希望跳过的字节数来跳过流中的部分数据:

```
public long skip(long n)
```

skip 方法实际上是使用 read 方法跳过字节,所以在 read 会发生阻塞的情况下它也会出现阻塞。它返回跳过的字节数,当遇到文件结尾时它返回 -1。

某些输入流允许用户在某一点上放置分类书签,这样用户就可以在以后返回这个位置。如果该流支持标志操作,那么 markSupported 方法返回 true:

```
public boolean markSupported( )
```

mark 方法在流的当前位置增加标志,这样用户以后就可以回到这个位置:

```
public synchronized void mark(int readLimit)
```

readLimit 参数设置了在该标志未被改变之前用户可以从该流中读取的字节的最大数量。换句话说,用户必须告诉流在忘记该标志之前应该允许用户读取多少字节。某些流可能需要分配支持标志操作的内存,并且 readLimit 参数告诉它们数组应该多大。

如果用户设置了一个标志,那么用户可以通过调用 reset 方法将当前位置定回该标志处:

```
public synchronized void reset( ) throws IOException
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

当使用完流时，用户应该使用 close 方法将其关闭：

```
public void close( ) throws IOException
```

大部分流在系统执行无用单元回收操作时自动地被关闭。在大部分操作系统中，用户可以同时打开的文件数是有限制的。因此当用户结束对流的使用时不要等待无用单元回收，用户应该主动关闭流以便马上释放系统资源。

27.2.2　OutputStream 类

与 InputStream 类相对的类是 OutputStream 类，该类为所有输出流提供了基本功能。表 27.2 列出了在 OutputStream 类中定义的方法和对这些方法的描述。

表 27.2　OutputStream 类的方法

方　　法	描　　述
write( )	向流中写入数据
flush( )	强迫缓冲区中的输出被写入
close( )	关闭流

输入流是数据的源，输出流是数据的拷贝。OutputStream 对象的最基本方法是 write 方法。

```
public abstract void write(int b) throws IOException
```

该方法向输出流写入一个字节。

```
public void write(byte[ ]bytes) throws IOException
```

该方法将字节数组的全部内容写入输出流。

```
public void write( byte[ ] bytes,int offset,int length) throws IOException
```

该方法将 bytes 数组中从 offset 位置开始的 length 个字节写入流。

根据流的类型，如果用户需要保证写在流上的数据被发送完毕，那么用户就可能需要冲洗该流。冲洗流并不会损坏流中任何信息，它只是要确保存储在内部缓冲区中的数据被写到与该流联接的输出设备上。为了清洗 输出流，用户可以调用 flush 方法：

```
public void flush( ) throws IOException
```

就像使用输流那样，当用户使用完输出流时，用户应该通过调用 close 方法将其关闭：

```
public void close( ) throws IOException
```

27.2.3　System.in 对象和 System.out 对象

为了支持标准输入和输出设备(通常分别是键盘和屏幕)，Java 定义了两个流对象。用户可以在自己的程序中直接使用这两个流对象，而不必自己亲自创建 System.in 对象(由 InputSystem 类实例化)允许用户的程序从键盘读取数据，而 System.out 对象(由 PrintSystem 类实例化)产生计算机屏幕输出。用户可以直接使用这两个流对象处理 Java 程序的标准输入和输

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

出。或者用户可以将它们用作创建其它流对象的基础。

程序清单 27.1 是一个 Java 应用程序示例，该应用程序接受用户的输入行并在屏幕上显示该行。图 27.1 显示了在 DOS 窗口中运行的该应用程序。

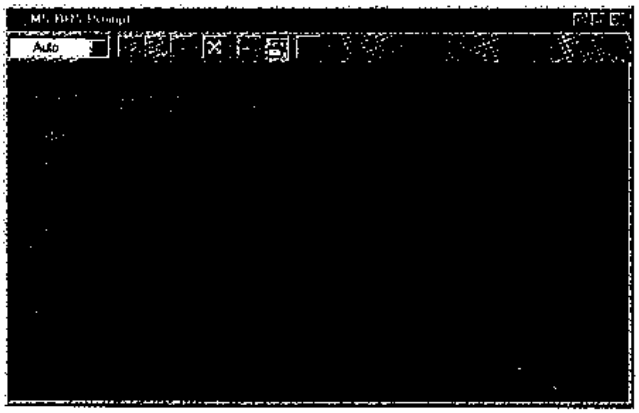


图 27.1 Java 的 System. 类提供了标准的输入和输出

```
import java.io.*;
class IOApp {
  public static void main(String args[]) {
    byte buffer[] = new byte[255];
    System.out.println("\nType a line of text:");
    try {
      System.in.read(buffer, 0, 255);
    }
    catch (Exception e) {
      String err = e.toString();
      System.out.println(err);
    }
    System.out.println("\nThe line you typed was:");
    try{
      String inputStr = new String(buffer, "Default");
      System.out.println(inputStr);
    }catch (UnsupportedEncodingException e){
      System.out.println("e:" + e);
    }
  }
}
```

27.2.4 PrintWriter 类

在程序清单 27.1 中用户可能注意到了名为 println( )的方法，该方法并不是 Output-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Stream 的一部分。为了在标准输出流上提供更灵活的输出，System 类从 PrintWrinter 类派生出它的外部输出流，PrintWriter 类提供文本打印输出。表 27.3 列出了 PrintWrinter 类的方法和对这些方法的描述。

表 27.3　PrintWrinter 类的基本方法

方　　法	描　　述
write( )	向流中写入数据
flush( )	冲洗流中的数据
checkError( )	冲洗流并返回发生的错误
print( )	按文本格式打印数据
println( )	按文本格式打印一行数据(有换行符跟在后面)
close( )	关闭流

就像使用流类里包含的其它方法一样，write( )、print( )、和 println( )方法都多次重载并且拥有多个版本。write( )方法可以写字符串，部分字符串、单个字符或整个字符数组，而 print( )方法和 println( )方法几乎可以在屏幕上显示各种类型的数据。各种方法签名如下所示：

```
void write(int c)
void write(char c[ ], int off, int len)
void write(String s)
void write(String s, int off, int len)
void print(Object obj)
void print(String s)
void print(char s[ ])
void print(char c)
void print(int i)
void print(long l)
void print(float f)
void print(double d)
void print(boolean b)
void println( )
void println(Object obj)
void println(String s)
void println(char s[ ])
void println(char c)
void println(int i)
void println(long l)
void println(float f)
void println(double d)
void println(boolean b)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

27.3 处理文件

既然用户已经在一定程度上了解了流类，那么用户就可以把这些知识用到工作上了。大概除了从键盘上接收数据和在屏幕上显示数据之外最常用的输入和输出就是文件输入和输出了。任何希望获得自身状态的程序（包括被编辑文件的状态）都必须能够加载和存储文件。Java提供了几个类，包括 File、FileDescriptor、RandomAccessFile、FileInputStream、FileOutputStream、FilePermission、FileReader 和 FileWriter，它们都是用于处理文件的。在本章的这一节里，用户可以考察这些类，看它们是如何工作的。

27.3.1 文件安全

当用户开始使用网络应用程序读取和写入磁盘时，必须考虑安全问题。因为 Java 语言是特意为创建基于网络的应用程序而设计的，所以安全性是非常重要的。没有使用者会担心他正在观看的网页是否有能力读取或写入磁盘。由于这个原因，Java 才被设计成允许使用者在他的兼容 Java 的浏览器中设置系统的安全性，并允许他决定哪些文件和目录可以被该浏览器访问或锁定哪些文件和目录。

在大部分情况下，使用者不会允许所有文件都能访问他的本地系统，这样可以保护他的系统免遭恶意侵人。事实上当前所有浏览器的缺省设置都不允许访问本地系统，甚至对文件系统的自愿访问也不允许。这种严密的安全性对小应用程序来说是至关重要的，因为小应用程序可以在使用者的背后将一些内容下载到使用者的系统上。如果人们害怕这种用法能够将本地系统向一些有恶意的人开放，那么就没有人愿意兼容 Java 的浏览器了。

Java 的独立应用程序则是完全不同的另一回事了。Java 应用程序与用户系统中的其它应用程序没有什么区别。它们不能像小应用程序那样自动地下载运行。本章的文件处理器就是具体的 Java 独立应用程序。

27.3.2 FileInputStream 类

如果用户的文件读取需求比较简单，那么用户可以使用 FileInputStream 类，该类是从 InputStream 中派生出来的简单输入流类。该类的所有方法都是从 FileInputStream 类继承来的。为了创建 FileInputStream 类的对象，用户可以调用它的构造器。下面显示了三个构造器：

```
FileInputStream(String name)
FileInputStream(File file)
FileInputStream(FileDescriptor fdObj)
```

第一个构造器使用给定的文件名 name 创建一个 FileInputStream 对象。第二个构造器使用 File 对象创建 FileInputStream 对象。第三个构造器使用 FileDescriptor 对象创建 FileInputStream 对象。

程序清单 27.2 是一个应用程序，该应用程序从磁盘上读取它自己的源代码并将代码显示在屏幕上。图 27.2 显示了该应用程序在 DOS 窗口中的输出。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。



图 27.2　FileApp 应用程序读并显示它自己的源代码

```
import java.io.*;
class FileApp {
  public static void main(String args[]) {
    byte buffer[] = new byte[2056];
    try {
      FileInputStream fileIn =
                                  new FileInputStream("fileapp.java");
      int bytes = fileIn.read(buffer, 0, 2056);
      try{
        String str = new String(buffer, 0, bytes, "Default");
        System.out.println(str);
      }catch (UnsupportedEncodingException e){
        System.out.println("The encoding \"Default\" was not found :" + e);
      }
    }
    catch (Exception e) {
      String err = e.toString();
      System.out.println(err);
    }
  }
}
```

27.3.3　使用 FileOutputStream 类

正像用户可能已经猜到的那样，与 FileInputStream 类相对应的类是 FileOutputStream 类，FileOutputStream 提供了基本的文件写入能力。除了从 OutputStream 类继承来的方法以外，FileOutputStream 类还有三个构造器，这三个构造器的签名如下所示：

```
FileOutputStream(String name)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

FileOutputStream(File file)

FileOutputStream(FileDescriptor fdObj)

第一个构造器使用给定的文件名 name 创建 FileOutputStream 对象，第二个构造器使用 File 对象创建 FileOutputStream 对象，第三个构造器使用 FileDescriptor 对象创建 FileOutputStream 对象。

程序清单 27.3 是一个 Java 应用程序，该应用程序从键盘读取一行文本并将该文本存储到文件中。当用户运行该应用程序时，需要输入一行文本并按下 Enter 键。然后在系统提示符下（在 DOS 窗口中）输入 TYPE LINE.TXT 并按下 ENTER 键，这时屏幕上会显示 LINE.TXT 文件中的文本；通过这条命令可以验证该应用程序的功能。图 27.3 显示了这个典型程序的运行情况。



图 27.3 FileApp2 应用程序将使用者的输入存储到文件中

```
import java.io.*;
class FileApp2
{
    public static void main(String args[])
    {
        byte buffer[] = new byte[80];
        try
        {
            System.out.println("\nEnter a line to be saved to disk:");
            int bytes = System.in.read(buffer);
            FileOutputStream fileOut = new FileOutputStream("line.txt");
            fileOut.write(buffer, 0, bytes);
        }
        catch (Exception e)
        {
            String err = e.toString();
            System.out.println(err);
        }
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

}

27.3.4　使用 File 类

如果用户需要获得关于文件的信息，那么用户需要创建一个 File 类的对象。File 允许用户向系统查询该文件的所有信息，比如文件名和文件最后修改时间等。用户也可以使用 File 类创建新的目录以及删除和重新命名文件。用户可以通过调用 File 类的构造器来创建 File 对象。File 类有三个构造器，它们的签名如下所示：

```
File(String path)
File(String path, String name)
File(File dir, String name)
```

第一个构造器使用完整的路径名和文件名组合(例如 C:\CLASSES\MYAPP.JAVA)创建 File 对象。第二个构造器使用路径名和文件名创建 File 对象。第三个构造器使用与另一个 File 对象关联的路径和文件名创建 File 对象。

File 类有一个完整的方法集，这些方法可以为应用程序提供许多文件处理选项。表 27.4 列出了这些方法和对它们的描述。

表 27.4　File 类方法

方　法	描　述
getName( )	获取文件名(返回字符串)
getPath( )	获取文件名的路径(返回字符串)
getAbsolutePath( )	获取文件的绝对路径(返回字符串)
getAbsoluteFile( )	获取文件的绝对路径(返回 File 对象)
getCanonicalPath( )	获取文件的典型路径(返回字符串)
getCanonicalFile ( )	获取文件的典型路径(返回 File 对象)
getParent( )	获取文件父目录(返回字符串)
getParentFile ( )	获取文件父目录(返回 File 对象)
exists( )	如果文件存在返回 true，否则返回 false
createNewFile( )	创建新文件，但只能在该文件尚不存在时方能创建(如果文件被创建则返回 true)
createTempFile (String pattern, File directory)	创建临时文件，使用 directory 作为目录名，使用 pattern 创建一个唯一的文件名(返回 File 对象)
createTempFile(String prefix)	创建临时文件。首先使用缺省的系统临时目录创建文件名，然后使用 prefix 创建一个唯一的文件名(返回 File 对象)
deleteOnExit( )	请求虚拟机退出时删除该文件。注意：只有当虚拟机成功地正常退出时才执行，意外的退出不执行
canWrite( )	如果文件能被写入返回 true
canRead( )	如果文件可读取返回 true
setReadOnly( )	设置文件使它只能被读取
isFile( )	如果文件有效返回 true

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

方　法	描　述
isDirectory( )	如果目录有效返回 true
isAbsolute( )	如果文件名是绝对的则返回 true
isHidden( )	测试文件看是否是隐含文件(如果是隐含文件则返回 true)
lastModified( )	返回文件最后一次被修改的时间,用从 1970 年 1 月 1 日格林威制时间 0 点 0 分 0 秒开始计算的毫秒数表示
setLastModified(long time)	设置文件最后一次被修改的时间。time 代表从 1970 年 1 月 1 日格林威制时间 0 点 0 分 0 秒开始计算的毫秒数
length( )	返回文件中的字节数
mkdir( )	创建一目录(如果创建成功则返回 true)
mkdirs( )	创建目录树(如果创建成功则返回 true)
renameTo(File dest)	将文件重新命名为指定的名字
list( )	获取目录中的文件列表(返回一个字符串数组)
list(FilenameFilter filter)	获取目录中与文件过滤器匹配的文件列表(返回一个字符串数组)
listFiles( )	获取目录中的文件列表(返回一个文件数组)
ListFiles(FilenameFilter filter)	获取目录中与文件过滤器匹配的文件列表(返回一个文件数组)
listRoots( )	获取当前系统的根目录列表。Windows 和 Macintosh 计算机的每个驱动器都有一个根,UNIX 只有一个根(/)(返回一个文件数组)
delete( )	删除文件(如果删除成功则返回 true)
hashCode( )	获取文件的一个散列码
equals( )	将该文件对象与另一个对象比较(如果它们相等则返回 true)
ToString	获取一个包含该文件路径的字符串
ToURL( )	返回一个与该文件对等的 URL 对象

目录操作　虽然 File 类的大部分方法都可以用于文件和目录,但是 list 方法只能用于目录:

```
public String [ ] list( )
```

list 方法返回包含在该目录中的所有文件的名字数组。用户也可以为 list 方法设置文件名过滤器,过滤器允许用户只选择部分文件名:

```
public String[ ] list(FilenameFilter filter)
```

FilenameFilter 接口定义了一个方法 accept,如果文件名包含在列表中 accept 返回 true;

```
public abstract boolean accept(File dir,String name)
```

程序清单 27.4 显示一个实现了文件名过滤器的对象,它只允许以 .java 为扩展名的文件通过。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.io.*;

// This class implements a filename filter that only allows
// files that end with .java

public class JavaFilter extends Object implements FilenameFilter
{
    public JavaFilter()
    {
    }

    public boolean accept(File dir, String name)
    {

// Only return true for accept if the file ends with .java
        return name.endsWith(".java");

    }
}
```

程序清单 27.5 显示了一个程序，该程序使用 JavaFilter 将当前目录中的所有 java 文件列出来。

```
import java.io.*;

public class ListJava extends Object
{
    public static void main(String[] args)
    {

// Create a File instance for the current directory
        File currDir = new File(".");

// Get a filtered list of the .java files in the current directory
        String[] javaFiles = currDir.list(new JavaFilter());

// Print out the contents of the javaFiles array
        for (int i=0; i < javaFiles.length; i++) {
            System.out.println(javaFiles[i]);
        }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }
}
```

创建临时文件 JDK1.2 的一个新特性是能够创建临时文件。在许多程序中,临时文件有各种各样的用途。为了创建临时文件,用户可以在 File 中使用 createTemporaryFile 方法。如程序清单 27.6 所示,创建临时文件需要传递一个用于代表该文件的命名方式的字符串,这个字符串包含该文件名的前几个字符(最少要包含三个)和英磅符号及扩展名。英磅符号位于标准键盘的数字键 4 上,必须保证它在该虚拟机实例中是唯一的。如果用户不提供该英镑符号,它将会被自动地追加在该字符串上;如果用户不提供扩展名,该处将被自动地使用 .tmp。

```
import java.io.*;
public class TestFile{
    public static void main(String args[]){
        try{
            File tempFile = File.createTempFile("test#que");
            FileOutputStream fout = new FileOutputStream(tempFile);
            PrintStream out = new PrintStream(fout);
            out.println("Place this test string in the temp file");
        }catch (IOException ioe){
            System.out.println("There was a problem creating/writing to the temp file");
            ioe.printStackTrace(System.err);
        }
    }
}
```

当用户运行程序清单 27.6 中的程序之后,将会在用户系统的临时目录中看到名为 test???? .que 的文件(这里的???? 实际上是数字)。

退出时删除文件 一种额外的特性通常用于临时文件,但是它也可被应用于任何文件。JDK1.2 中增加了 deleteOnExit( )方法,该方法可以安排某个文件在虚拟机退出时被删除。这个特性很有用,当用户在自己的程序中创建一个文件,但并不希望它在程序退出后仍然存在,在这种情况下用户就可以使用 deleteOnExit( )方法。使用这个方法需要警告用户两点:第一,一但文件被安排在程序退出时删除,那么用户就不能取消这种安排。第二,deleteOnExit( )方法只有当虚拟机正常退出时才有效,比如由 System.exit(0)退出。如果系统崩溃或其它形式的不规则崩溃,那么虚拟机就无法删除该文件。

程序清单 27.7 显示了如何将程序清单 27.6 中的 TestFile 修改成能够在系统退出时删除临时文件的程序。注意 TestFile2 尚未退出之前用户可以在当前目录有一个文件,但是当 TestFile2 退出时这个文件就不见了。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
import java.io.*;
public class TestFile2{
    public static void main(String args[]){
        try{
            File tempFile = File.createTempFile("test#que");
            FileOutputStream fout = new FileOutputStream(tempFile);
            PrintStream out = new PrintStream(fout);
            out.println("Place this test string in the temp file");
            tempFile.deleteOnExit();
        }catch (IOException ioe){
            System.out.println("There was a problem creating/writing to the temp file_
            ioe.printStackTrace(System.err);
        }

        System.out.println("Until you hit 'nter' there is a temp file on the system");_
        try{
            System.in.read();
        }catch (IOException ioe){
        }
        System.exit(0);
    }
}
```

27.3.5　RandomAccessFile 类

到现在，用户可能认为 Java 的文件处理能力是被分散在许多不同的类中，想获得基本的读取、写入和管理文件的功能是有困难的。但是 Java 的缔造者是超前于用户的。他们创建了 RandomAccessFile 类，当用户真正需要严格地处理文件时，就可以使用这个类。通过这个类用户可以随自己所愿地使用文件做任何事情。

用户可以通过调用 RandomAccessFile 类的两个构造器中的任何一个来创建 RandomAccessFile 对象，RandomAccessFile 类的构造器签名如下所示：

```
RandomAccessFile(String name, String mode)
RandomAccessFile(File file, String mode)
```

第一个构造器用一个包含文件名的字符串和另一个包含访问方式的字符串(r 为读方式，rw 是读写方式)创建 RandomAccessFile 对象。第二个构造器用一个 File 对象和方式字符串创建 RandomAccessFile 对象。当用户创建了 RandomAccessFile 对象以后，用户就可以调用该对象的方法处理文件了。表 27.5 列出了这些方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 27.5 RandomAccessFile 类的方法

方 法	描 述
close( )	关闭文件
getFD( )	获取文件的 FileDescriptor
GetFilePointer( )	获取文件指针的位置
length( )	获取文件的长度
read( )	从文件中读取数据
ReadBoolean( )	从文件中读取一布尔值
ReadByte( )	从文件中读取一个字节
ReadChar( )	从文件中读取一个字符
ReadDouble( )	从文件中读取一个双精度浮点值
ReadFloat( )	从文件中读取一个单精度浮点值
ReadFully( )	将数据读入数组,完全填满该数组
ReadInt( )	从文件中读取一个 int 值
ReadLine( )	从文件中读取一个文本行
Readlong( )	从文件中读取一个长型 int 值
ReadShort( )	从文件中读取一个短型 int 值
ReadUnsignedByte( )	从文件中读取一个无符号字节
ReadUnsignedShort( )	从文件中读取一个无符号短型 int 值
ReadUTF( )	从文件中读取一个 UTF 字符串
seek( )	定位文件指针在文件中的位置
skipBytes( )	在文件中跳过给定数量的字节
write( )	向文件写数据
writeBoolean( )	向文件写一个布尔值
writeByte( )	向文件写入一个字节
writeBytes( )	向文件写入一个字符串
writeChar( )	向文件写入一个字符
writeChars( )	向文件写入一个作为字符数据的字符串
writeDouble( )	向文件写入一个双精度浮点值
writeFloat( )	向文件写入一个单精度浮点值
writeInt( )	向文件写入一个 int 值
writeLong( )	向文件写入一个长型 int 值
writeShort( )	向文件写入一个短型 int 值
writeUTF( )	写入一个 UTF 字符串

程序清单 27.8 是一个 Java 应用程序,该应用程序使用 RamdomAccessFile 对象显示了它自己的源代码。图 27.4 显示了一个典型程序的运行。

```
import java.io.*;
class FileApp3
{
    public static void main(String args[])
    {
        try
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        RandomAccessFile file =
            new RandomAccessFile("fileapp3.java","r");
        long filePointer = 0;
        long length = file.length();
        while (filePointer < length)
        {
            String s = file.readLine();
            System.out.println(s);
            filePointer = file.getFilePointer();
        }
    }
    catch (Exception e)
    {
        String err = e.toString();
        System.out.println(err);
    }
}
```



图 27.4　FileApp3 应用程序可以读取和显示它自己的源代码

27.4　使用管道

Java 普通的流和文件处理与其它计算机语言中的流和文件处理并非完全不同。Java 的流类提供了各种流处理能力。另外，Java 还支持管道，管道是一种数据流形式，用户可能没有使用它的经验。管道是不同线程之间直接传输数据的基本手段。一个线程通过它的输出管道发送数据，另一个线程从自己的输入管道中读取数据。通过使用管道，用户可以在不同的线程之间实现数据共享，不必借助其它的东西，比如临时文件。

27.4.1　PipedInputStream 类和 PipedOutputStream 类

正像用户可能已经猜到的那样，Java 为处理管道提供了两个类。第一个类是 PipedInput-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Stream，PipedInputStream 描述了管道的输入；第二个类是 PipedOutputStream 类，PipedOutputStream 类描述了管道的输出。两个类一起工作可以提供一个有管道的数据流，就像常规的管道为水流提供良好的流动环境一样。如果用户关闭常规管道的一端，那么水的流就会停止。有管道的流也是这样。如果用户不同时拥有输入流和输出流，就相当于封住了数据管道的一端或两端。

为了创建有管道的流，用户需要首先创建一个 PipedOutputStream 类的一个对象。然后再创建 PipedInputStream 类的一个对象，传给它一个对有管道的输出流的引用，像这样：

```
pipeOut = new PipedOutputStream( );
pipeIn = new PipedInputStream(pipeOut);
```

通过传给 PipedInputStream 对象一个对输出管道的引用，用户已经有效地把输入和输出连接到一个流上了，通过这个流数据就可以沿着一个方向流动了。泵到管道的输出端的数据可以被访问该管道的输入端的另一个线程接收到，如图 27.5 所示。

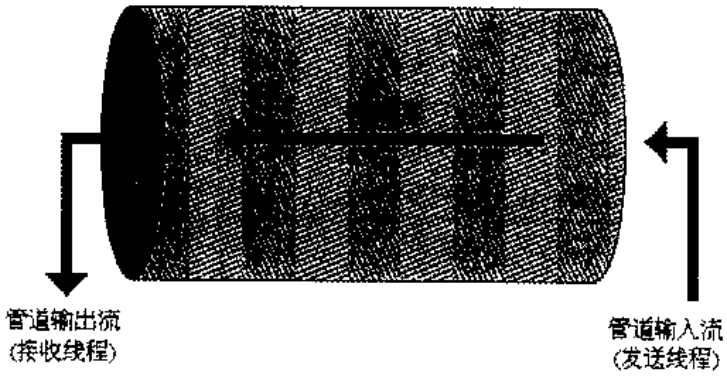


图 27.5 输出流和输入流扮演着单向管道的两端

> **注意**：它看上去可能有些奇怪，管道的输出端泵入数据，而输入端是数据流出的端。用户必须考虑针对使用该管道的线程来说，不是从管道自身的角度说。既提供数据的线程将它的输出送向有管道的输出流，而输入数据的线程从有管道的输入流中拿取数据。

当用户创建了管道以后，用户就可以读取和写入数据了，就像使用常规的文件那样。在接下来的部分中用户可以看一下运作中的管道。

27.4.2 PipeApp 应用程序

程序清单 27.9 和 27.10 以及 27.11 都是使用管处处理数据的名为 PipeApp 的应用程序。该应用程序有三个线程：一个主线程和两个由主线程启动的辅助线程。该应用程序使用了一个包含的数据全部是 X 的文件，在使用管道传递数据之前，首先将数据全部改成 Y，然后将这些数据再全部改成 Z，最后该程序在屏幕上显示修改过的数据。注意在这个过程中除了输入文件以外没有附加的文件被创建过。所有数据都是使用管道来操作的。图 27.6 显示了程序的运行结果。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

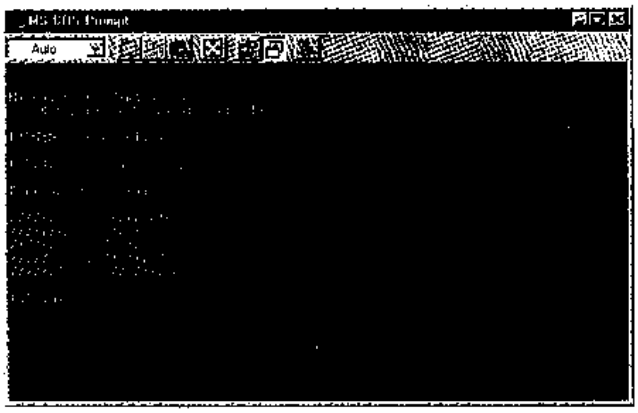


图 27.6　PipeApp 应用程序使用管道来实现三个线程之间的数据共享

程序清单 27.9　PipeApp.java——PipeApp 应用程序的主程序

```
import java.io.*;
public class PipeApp {
  public static void main(String[] args) {
    PipeApp pipeApp = new PipeApp();
    try {
      FileInputStream XFileIn = new FileInputStream("input.txt");
      InputStream YInPipe = pipeApp.changeToY(XFileIn);
      InputStream ZInPipe = pipeApp.changeToZ(YInPipe);
      System.out.println();
      System.out.println("Here are the results:");
      System.out.println();
      BufferedReader reader = new BufferedReader(new InputStreamReader(ZInPipe));
      String str = reader.readLine();
      while (str != null) {
        System.out.println(str);
        str = reader.readLine();
      }
      reader.close();
    }
    catch (Exception e) {
      System.out.println(e.toString());
    }
  }
  public InputStream changeToY(InputStream inputStream) {
    try {
      BufferedReader XFileIn = new BufferedReader(new InputStreamReader(inputStream));
      PipedOutputStream pipeOut = new PipedOutputStream();
      PipedInputStream pipeIn = new PipedInputStream(pipeOut);
      PrintWriter printWriter = new PrintWriter(pipeOut);
      YThread yThread = new YThread(XFileIn, printWriter);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            yThread.start();
            return pipeIn;
        }
        catch (Exception e) {
            System.out.println(e.toString());
        }
        return null;
    }
    public InputStream changeToZ(InputStream inputStream) {
        try {
            BufferedReader YFileIn = new BufferedReader(new InputStreamReader(inputStream));
            PipedOutputStream pipeOut2 = new PipedOutputStream();
            PipedInputStream pipeIn2 = new PipedInputStream(pipeOut2);
            PrintWriter printWriter2 = new PrintWriter(pipeOut2);
            ZThread zThread = new ZThread(YFileIn, printWriter2);
            zThread.start();
            return pipeIn2;
        }
        catch (Exception e) {
            System.out.println(e.toString());
        }
        return null;
    }
}
```

```
import java.io.*;
class YThread extends Thread {
    BufferedReader XFileIn;
    PrintWriter printWriter;
    YThread(BufferedReader XFileIn, PrintWriter printWriter) {
        this.XFileIn = XFileIn;
        this.printWriter = printWriter;
    }
    public void run() {
        try {
            String XString = XFileIn.readLine();
            while (XString != null) {
                String YString = XString.replace('X','Y');
                printWriter.println(YString);
                printWriter.flush();
                XString = XFileIn.readLine();
            }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            printWriter.close();
        }
        catch (IOException e) {
            System.out.println(e.toString());
        }
    }
}
```

程序清单 27.11　ZThread.java——将数据全部改变成 Z 的线程

```
import java.io.*;
class ZThread extends Thread {
    BufferedReader YFileIn;
    PrintWriter printWriter;
    ZThread(BufferedReader YFileIn, PrintWriter printWriter) {
        this.YFileIn = YFileIn;
        this.printWriter = printWriter;
    }
    public void run() {
        try {
            String YString = YFileIn.readLine();
            while (YString != null) {
                String ZString = YString.replace('Y','Z');
                printWriter.println(ZString);
                printWriter.flush();
                YString = YFileIn.readLine();
            }
            printWriter.close();
        }
        catch (IOException e) {
            System.out.println(e.toString());
        }
    }
}
```

27.4.3　探究 main( )方法

看 PipeApp 应用程序运行和理解它为什么这样运行是完全不同的两码事，在本部分中，用户可以逐行地考察该程序，看它们是做什么用的。PipeApp.java 文件是一个主程序线程，所以用户有必要从该应用程序开始考察。该应用程序包含三个方法：main( )方法、changeToY( )方法和 changeToZ( )方法，其中 main( )方法是所有应用必须拥有的方法，changeToY( )方法和 changeToZ( )方法启动了两个附加的线程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在 main( )方法中,该程序首先为自己创建了应用程序对象:

```
PipeApp pipeApp = new PipeApp( );
```

这对调用 changeToY( )方法和 changeToZ( )方法是必要的。这两个方法在应用程序被创建之前它们都是不存在的。实现这种调用的一个办法是将所有类的方法都设置成静态的,而不仅仅 main( )方法是静态的。然后用户就可以不必创建类的对象而直接调用这些方法。

当创建完应用程序对象后,应该设置一个 try 程序块,流要求用户的代码能够捕捉 IOException 异常。在 try 块中程序为源文本文件创建一个输入流:

```
FileInputStream XFileIn = new FileInputStream("input.txt");
```

这个新输入流被传递给 changeToY( )方法,这样下一个线程就可以读取该文件了。

```
InputStream YInPipe = pipeApp.changeToY(XFileIn);
```

changeToY( )方法创建了将输入数据全部修改成 Y 的线程(用户将会在后面的"探究 changeToY( )方法"中看到这个方法是如何工作的)从该线程中返回输入管道。下一个线可以使用该输入管道访问由第一个线程创建的数据。因此该输入管道做为一个参数传递给了 changeToZ( )方法:

```
InputStream ZInPipe = pipApp.changeToZ(YInPipe)
```

changeToZ( )方法启动了将数据由全部为 Y 修改成全部为 Z。主程序使用由 changeToZ( )返回的输入管道访问被修改过的数据并将它打印在屏幕上。

当程序获得 ZInPipe 管道输入流后,程序就在屏幕上打印一则信息:

```
System.out.println( );
System.out.println("Here are the results:");
System.out.println( );
```

然后,程序将管道输入流映射到 BufferedReader 对象上,这样程序就可以使用 readLine( )方法读取数据了:

```
BufferedReader reader = new Buffereder(new InputStreamReader(ZInPipe));
```

当输入流被创建以后,程序就可以逐行读取数据并在屏幕上将它们显示出来。如程序清单 27.12 所示。

```
String str = reader.readLine();
while (str ! = null)
{
    System.out.println(str);
    str = inputStream.readLine();
}
```

最后,当显示完毕数据时,程序将输入流关闭:

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
reader.close( );
```

27.4.4　探究 changeToY( )方法

changeToY( )方法是在本程序中用户看到管道起作用的第一个地方。像 main( )方法一样,为了捕捉 IOException 异常,changeToY( )方法的大部分操作是在 try 程序块中进行的。该方法首先将作为自己唯一参数传递过来的输入源映射成 DataIputStream 对象。这使得程序能够使用 readLine( )方法从流中读取数据:

```
Buffered XFileIn = new Buffered(new InputStreamReader (inputStream));
```

接着,changeToY( )方法开始创建输出管道和输入管道:

```
PipedOutputStream pipeOut = new PipedOutputStream( );
PipedOutputStream pipeIn = new PipedOutput(pipeOut);
```

然后,为了能够使用 println( )方法向管道输出文本行,本程序将输出管道映射到 PrintStream 对象上:

```
PrintWriter printWriter = new PrintWriter( pipeOut);
```

到现在为止,changeToY( )方法共创建了四个流:

- 第一个流是 XFileIn,该流描述了将要从文件中读取的数据。
- 第二个流是 pipeOut,第三个流是 printWriter,这两个流用户可以认为是一个流,这两个流是管道的输出端,新线程将向它输出数据。
- 第四个流是 pipeIn,该流是管道的输入端,下一个线程将由此输入它的数据。

图 27.7 说明了这个情况。

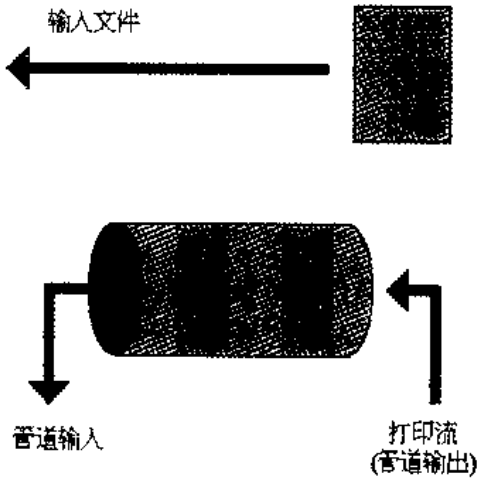


图 27.7　changeToY( )方法创建的流

现在程序就可以创建将数据 X 改成 Y 的线程了。该线程是 YTread 类的对象,它的构造器使用输入文件(XFileIn)和输出文件(现在是 printWriter)作为参数:

```
YTread yTread = new YTread(XFileIn,printWrter);
```

当创建完线程这后,程序就可以启动该线程了:

```
yTread.start( );
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

正如用户很快就可以看到的那样，YTread 线程从 XFileIn 中读取数据，接着将数据由 X 改成 Y，然后再将结果输出给 printWriter，既管道的输出端。因为管道的输出端是与输入端(pineIn)是连接在一起的。所以输入端包含着要改变成 Y 的数据。程序从 changetTOY( )中返回管道的输入端以便该端能被 changeTOY( )方法用作输入。图 27.8 显示了该传输链的 changeTOY 部分。

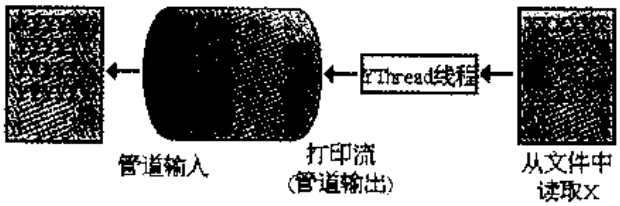


图 27.8 changeToY( )方法读入 X 并把 Y 送入管道中

27.4.5 探究 changeToZ( )方法

changetToZ( )方法的工作过程类似于 changetToY( )方法。但是，因为每个方法访问流的方式对于理解 PipeApp 应用程序是很重要的，所以用户也应该逐行地考察 changetToZ( )方法。changetToZ( )方法是以映射它的输入流开始的，该流是从 changetToY( )方法返回来的管道输入端，在 changetToZ( )方法中它被映射到了 BufferedReader 对象上，这样，程序就可以使用 readLine( )方法从流中读取数据了：

```
Buffered YFileIn = new BufferedReader(new InputStream Reader(inputStream));
```

然后，程序开始创建新的管道：

```
PipedOutPutStream pipeOut2 = new PipedOutputStream( );
PipedOutPutStream pipeIn2 = new PipedOutputStream (pipeOut2);
```

新的管道给数据定下了从第三个线程(main 线程也计算在内)到主程序的传输路线。

当创建完管道以后，程序将输出端映射到 PrintStream 对象上，这样使用 println( )方法就可以将数据送入管道了：

```
PrintWriter printWriter2 = new PrintWriter(pipeOut2);
```

接下来，程序从 ZThread 类使用 ZThread 类创建一个线程，ZThread 类的构造器使用的两个参数是 changeToY( )方法创建的输入管道和另一个新的管道：

```
ZThread Zthread = new ZThread(YFileIn,printStream2);
```

接下来的语句启动该线程：

```
ZThread.start();
```

ZThread 线程从 changeToY( )方法创建的输入管道读取由 YThread 线程提供的数据，然后，将数据修改成 Z，最后将数据输出给名为 printWriter2 的输出管道。changeToZ 方法返回该管道的输入端，主程序在屏幕上打印该流中的内容。图 27.9 说明了流的概要。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

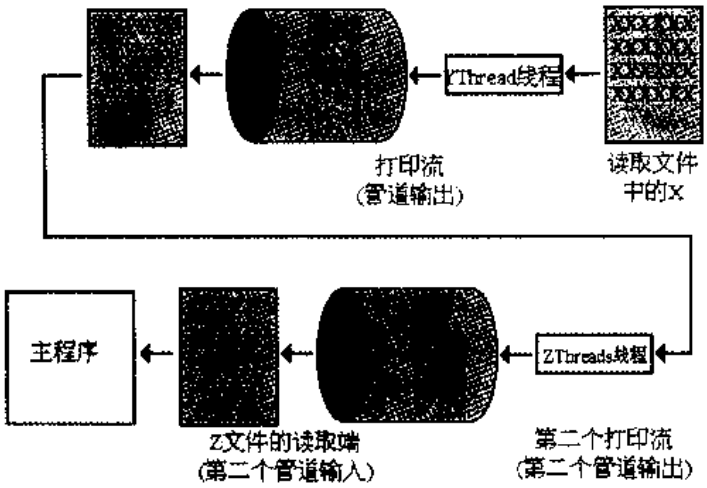


图 27.9　数据由 Y 变成 Z 要经历一段很长的路程

27.4.6　探究 YThread 类

现在用户应该已经基本了解了管道是如何工作的。如果说还有疑惑，那可能就是为管道服务的辅助线程序 YThread 和 Zthread 了。因为这两个线路的工作方式几乎是一致的，所以用户只要考察一个 YThread 线程就可以了。

YThread 的构造器接受两个参数：输入文件和第一个管道的输出端。构造器将这两个参数当作数据成员来保存：

```
this.XfileIn = XFileIn;
this.printWriter = printWriter;
```

有了流以后，该线程就可以开始在 run( )方法中处理数据了。首先。线程从输入文件中读取一行数据；然后进入处理文件中所有数据的 while 循环。在循 环开始之前读取一行数据可以保证 XString 不为空，XString 为空将结束循环的执行：

```
String XString = XFileIn.readLine( );
while(XString ! = null)
```

在该循环中，线程首先将新读入的数据修改成 Y：

```
String YString = XString.replace('X','Y');
```

然后，将修改过的数据输出到管道的输出端：

```
printWriter.println(YString);
printWriter.flush( );
```

冲洗流非常重要，它可以保证缓冲区里的数据全部被输出到管道中。再后，线程为循环中下一次迭代读取另一行数据：

```
XString = SFileIn.readLine( );
```

最后，当循环结束时，线程将关闭输出流：

```
printWriter.close( );
```

这就是全过程。概括起来说，该线程只是从输入文件中读入数据行，之后将数据行中的字

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

符改成 Y，然后将修改过的数据送入管道。下一个线程从管道中获取这些数据。

ZThread 线程以几乎相同的方式工作，除了它的输入是由 YThread 线程向它输出数据以外。最后，当程序读取文本行并在屏幕上显示文本时由 ZThread 的管道输入端提供数据。

27.4.7 过滤流

流的最强大功能之一是允许用户将一个流链接到另一个流的尾部。例如，基本输入流为读取字节提供了 read 方法，如果用户希望读取字符串和整数，那么用户就可以将一个特殊的数据输入流附加到某个输入流上，这样就可以马上拥有读取字符串和整数的方法；甚至于读取浮点数的方法。FilterInputStream 类和 FilterOutputStream 类提供了链接在一起的能力。但是它们并不增加任何新的方法。它们的最大贡献是它们被"连接"到另一个流上。FilterInputStream 类的构造器使用 InputStream 对象作为参数，FilterOutputStream 类的构造器使用 OutputStream 对象作为参数：

```
public FilterInputStream(InputStream in)
public FilterInputStream(OutputStream out)
```

因为这两个类本身就是 InputStream 的实例，所以它们可以用作其它过滤器的参数，这可以使用户能够创建一个很长的输入和输出过滤器链。

在本章后面将要讨论的 DataInputSteam 类是一个非常有用的过滤器，它允许用户从输入流中读取整数、字符串和其它简单类型的数据。另外 LineNumberInputStream 过滤器自动地数用户读取输入的行数。当需要计算行数时，用户可以把这些过滤器链接在一起：

```
LineNumberInputStream lineCount = new LineNumberInput Stream(System.in);
DataInputStream dataIn = new DataInputStream(lineCacnt);
```

27.5 缓冲区里的流

缓冲区里的流通过减少对磁盘的读写来加快用户程序的运行速度。假设用户的程序每次只要写一个字节，那么用户可能不希望每次都去写磁盘。用户可能会喜欢将这些字节积累成一个大的块后整批地写入磁盘。BufferedInputStream 类和 BufferedOutputStream 类提供了这中功能。当用户创建它们时，用户可以为它们提供缓冲区的尺寸：

```
public BufferedInputStream(InputStream in)
public BufferedInputStream(InputStream in,int bufferSize)
public BufferedInputStream(OutputStream out)
public BufferedInputStream(OutputStream out,int buffer Size)
```

BufferedInputStream 类在每次调用 read 时总是试图尽可能多地将数据读入它的缓冲区中；BufferedOutputStream 类当缓冲区被填满时或当 flush 方法被调用时才调用 write 方法。

27.6 数据流

DataInputStream 和 DataOutputaStream 过滤器是 java.io 包中最有用途的两个过滤器。它们允许用户按照机器无关风格读取 Java 原始类型数据。当用户希望在一个机器上写文件，而

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在另一个使用不同 CPU 结构的机器上读取该文件时这一点就显得很重要的了。例如，在使用 Intel 芯片的 PC 机和使用 Sparc 芯片的工作站之间传递数据的最一般的困难之一是这两种 CPU 芯片存放整数的方式不同。存放在 Sun 工作站上的 012345678 将被 PC 机解释为 078564312。幸运的是 DataInputStream 和 DataOutputStream 类自动地处理任何必要的转换。

27.6.1　DataInput 接口

DataInputStream 实现了 DataInput 接口。该接口定义了读取 Java 原始数据类型的方法以及其它几个方法。

DataInput 读取原始数据类型的方法有：

```
public boolean readBoolean( ) throws IOException, EOFException
public boolean readByte( ) throws IOException, EOFException
public boolean readChar( ) throws IOException, EOFException
public boolean readShort( ) throws IOException, EOFException
public boolean readInt( ) throws IOException, EOFException
public long readLong( ) throws IOException,EOFException
public float readFloat( ) throws IOException,EOFException
public double readDouble( ) throws IOException, EOFException
```

有时用户可能需要读取无符号的字节或整型数(16 位整数)，这时用户就可以使用 readUnsignedByte 方法和 readUnsignedShort 方法：

```
public int readUnsignedByte( ) throws IOException, EOFException
public int readUnsignedShort( ) throws IOException, EOFException
```

用户可能期望 DataInput 接口包含一个 readString 方法。它有，但不叫 readString。读字符串的方法叫做 readUF。UTF 代表统一码传输格式(既英文 UnicodeTransmission Format 的缩写)，它是一种用 16 位统一码值编码的特殊格式。UTF 假设统一码值大部分时候高 8 位为 0 做此优化：

```
public String readUTF throws IOException,EOFException
```

当用户试图在 InputStream 类中使用标准的 read 方法向某个数组读入固定数量的字节时，用户可能必须多次地调用 read 方法，因它在读完这些字节前就可能返回。当用户通过网络传输数据时尤其是这样，readFully 方法明确地等待所有用户请求所有字符：

```
public void readFully(byte[ ] bytes) throws IOException,EOFException
public void readFully(byte[ ] bytes ,int offset, int length) throws IOException, EOFException
```

注意，readFully 方法不像标准的 read 方法那样，它不返回字节数。这是因为用户应该已经知道将要读取多少字节，在方法的第一个版本中字节数是由数组的尺寸决定的，在第二个版本中是由 length 决定的。skipBytes 执行的功能类似于 readFully 方法，它在返回之前一直要等到希望跳过的字节全部被跳过。事实上，如果将该方法称为"skipBytes Fully"更恰当。

```
public int skipBytes(int numBytes)
```

27.6.2　DataOutput 接口

DataOutput 接口定义了与 DataInput 接口定义的方法相应的一些方法。由 DataOutput 定

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

义的方法如下所示：

```
public void writeBoolean(boolean b) throws IOException
public void writeByte(int b)
public void writeChar(int c) throws IOException
public void writeShort(int c) throws IOException
public void writeInt(int i) throws IOException
public void writeLong(int l) throws IOException
public void writeFloat(float f) throws IOException
public void writeDouble(double c) throws IOException
public void writeUTF(Sthring s)) throws IOException
```

用户也可以使用 writeBytes 方法和 writeChars 方法将字符串写成一系列字节或字符：

```
public void writeBytes(String s) throws IOException
public void writeChars(String s) throws IOException
```

27.6.3 DataInputStream 类和 DataOutputStream 类

DataInputStream 类和 DataOutputStream 类就是实现 DataInput 接口和 DataOutput 接口的流过滤器。它们的构造器是典型的流过滤器构造器，该构造器只使用要过滤的流作为参数：

```
public DataInputStream(InputStream in)
public DataInputStream(OutputStream out)
```

27.7 字节数组流

用户不必总是使用流写文件，用户也可以使用 ByteArrayInputStream 类和 ByteOutputString 类写入和读取字节数组。这些流不是过滤器流，它们是输入输出流。

当用户创建 ByteArrayInputStream 时，用户必须提供一个字节数组，这个数组将用作从流中读入字节的源。

```
public ByteArrayInputStream(byte[ ] bytes)
```

该方法使用 bytes 的全部内容作为流的数据创建一个字节输入流。

```
pubic ByteArrayInputStream(byte[ ] bytes, int offset, int length)
```

该方法创建一个字节输入流，该流从数组 bytes 的 offset 位置开始最多读取 length 个字节。

ByteArrayInputStream 类有一个可随着存放的数据继续增长的字节数组。ByteArrayInputStream 类的构造器可以选择使用初始尺寸作为参数，该参数决定存放写入流中字节的数量：

```
public ByteArrayInputStream( )
public ByteArrayInputStream(int initialSize)
```

当用户将数据写入 ByteArrayInputStream 以后，用户可以通过调用 toByteArray 方法将流的内容转换成一个字节数组：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public synchronized byte[ ] toByteArray( )
```

size 方法返回到目前为止写入流的字节总数：

```
public int size( )
```

27.8　字符数组流

Java 和像 C 这样 的语言之间的一个差别是 Java 将字符当作 16 位值来处理，而不是一般的 8 位值。不幸的是，在 Java1.0 中字节数组流没有 16 位的版本。这促使了程序员当希望使用字节流时将字符当作 8 位来处理。幸运的是在 Java1.1 中包含了字符数组流，但是它们不再称作流。Java1.1 引进了一个新的流类型叫做“读取器”或“写入器。具体叫哪一个名称取决于它是输入流还是输出流。ByteArrayInputStream 的字符版本叫做 CharArrayReader，CharArrayWriter 的功能类似于 ByteArrayInputStream。

CharArrayReader 类和 CharArrayWriter 的功能几乎能与 ByteArrayInputStream 和 ByteArrayOutputStream 一致，它们包含相同的方法，差别只是字符数组流使用字符值，而字节数组流使用字节值。charArrayOutputStream 的构造器如下所示：

```
public CharArrayReader(char[ ] buf)
public CharArrayReader(char[ ] buf,int offset, int length)
public CharArrayWriter( )
public CharArrayWriter(int size)
```

27.9　字节和字符之间的转换

因为许多系统都将字符预处理为 8 位值，所以用户经常会遇到这种情况，既需要将 8 位值转换为 16 位的 Java 字符。Java 字符之所以采用了 16 位格式是为了支持统一码标准。统一码标准支持各种国际字符集。目前许多程序员只使用 8 位值并假设高 8 位为 0。不幸的是这并不能满足各种不同的情况。Java1.1 增加了一个标准的转换机制，它既可以将字节转换成字符，也可以将字符转换成字节。但这并不意味着是标准转换。用户仍然需要设计自己要用的转换。当设计完转换之后就等告诉了输入流和输出流一个向字符转换的标准方式。

Reader 类和 Writer 类描述了与字符操作紧密联系的特殊输入流和输出流。这两个类自动地执行字节与字符之间的转换。对于各种核心输入流和输出流，Reader 和 Writer 版本都是负责字符操作的。下面的读取器和写入器版本与它们的流对应物是几乎一致的，除了它们使用 Reader 作参数而不是 InputStream，或者使用 Writer 作参数而不使用 OutputStream：BufferedReader、BufferedWriter、FileReader、FileWriter、FilterReader、FilterWriter、PipedReader、PipedWriter，PrintWrinter 和 PushbackReader。

InputStreamReader 类和 OutputStreamWriter 类在流和 Reader/Writer 类之间提供一座桥梁。Inputstream 使用 Inputstream 作参数并为它提供一个 Reader 接口。构造器可以选择使用编码名，即用于将字节转换成字符的字符编码名：

```
public IuputStreamReader(InputStream in)
public IuputStreamReader(InputStream in,String encoding)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

类似地，OutputStreamWriter 类为 OutputStream 提供了 Writer 接口，并且可以使用可选的编码参数：

```
public OutputStreamWriter(OutputStream out)
public OutputStreamWriter(OutputStream outString encoding)
```

Reader 类和 Writer 类允许用户在自己的程序中无缝地支持国际字符集。如果用户的程序将要读写的数据可能是另一国语言，那么用户就应该使用 Reader/Writer 对象而不是流。

27.10 StringBufferInputStream

StringBufferInputStream 是 ByteArrayInputStream 的近亲。两者间唯一的差别是 StringBufferInputStream 的构造器使用字符串作为流字符的源而不是使用字节数组：

```
public StringBufferInputStream(String str)
```

27.11 对象流

当 Sun 公司在 Java 中增加远程方法调用(RMI)时，它也增加了流动任意对象的能力。ObjectInput 接口和 ObjectOutput 接口定义了读写任意对象的方法，以相同的方式 DataInput 和 DataOutput 定义了读写原始类型的方法。事实上，ObjectInput 接口和 ObjectOutput 接口都扩展了 DataInput 接口和 DataOutput 接口。而 ObjectInput 接口唯一增加了一个输入方法：

```
public abstract Object readObject( ) throws ClassNotFoundException, IOException
```

类似地 ObjectOutput 接口唯一地增加了一个输出方法：

```
public abstract void writeObject(Object obj) throws IOException
```

ObjectOutputStream 实现的流过滤器允许用户向流写入任意对象和任意原始类型的功能。像某些流过滤器一样，用户可以通过传递 OutputStream 来创建 ObjectOutputStream：

```
public OutputStream(OutputStream outStream)
```

用户可以使用 writeObject 方法在流中写入任何对象：

```
public final void writeObject(Object ob) throws ClassMismatchException, MethodMissingException, IOException
```

因为 ObjectOutputStream 类是 DataOutputStream 类的子类，所以用户也可以使用来自于 DataOutput 接口的任何方法，比如 WriteInt 或 WriteUTF。

程序清单 27.13 显示了一个程序，该程序使用 writeObject 将数据和散列表传递给文件。

```
import java.io.*;
import java.util.*;

// This class writes out a date object and a hash table object
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// to a file called "writeme" using an ObjectOutputStream.

public class WriteObject extends Object
{
    public static void main(String[] args)
    {

// Create a hash table with a few entries

        Hashtable writeHash = new Hashtable();
        writeHash.put("Leader","Moe");
        writeHash.put("Lieutenant","Larry");
        writeHash.put("Stooge","Curly");

        try {

// Create an output stream to a file called "writeme"
            FileOutputStream fileOut =
                new FileOutputStream("writeme");

// Open an output stream filter on the file stream
            ObjectOutputStream objOut = new ObjectOutputStream(fileOut);

// Write out the current date and the hash table
            objOut.writeObject(new Date());
            objOut.writeObject(writeHash);

// Close the stream
            objOut.close();

        } catch (Exception writeErr) {
// Dump out any error information
            writeErr.printStackTrace();
        }
    }
}
```

正如用户可能已经猜到的那样，ObjectInputStream 为 ObjectInput 接口实现了流过滤器。用户可以通过向构造器传递用户希望过滤的输入流来创建 ObjectInputStream 对象：

```
public ObjectInputStream(InputStream inStream)
```

readObject 方法从输入流中读取一个对象：

```
public final Object readObject( ) throws MethodMissingException, ClassMismatchException, ClassNot-
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

FoundException, StreamCorruptedException, IOException

用户也可以在 ObjectInputStream 上使用来自于 DataInput 接口的任何方法。

程序清单 27.14 显示了使用 Readobject 读对象的程序,此对象由在程序清单 27.13 中的例子写到"writeme"文件中。

程序清单 27.14 ReadObject.java 的源代码

```
import java.io.*;
import java.util.*;

// This class opens up the file "writeme" and reads two
// objects from it. It makes no assumptions about the
// types of the objects, it just prints them out.

public class ReadObject extends Object
{
    public static void main(String[] args)
    {
        try {

// Open an input stream to the file "writeme"
            FileInputStream fileIn = new FileInputStream("writeme");

// Create an ObjectInput filter on the stream
            ObjectInputStream objIn = new ObjectInputStream(fileIn);

// Read in the first object and print it
            Object ob1 = objIn.readObject();
            System.out.println(ob1);

// Read in the second object and print it
            Object ob2 = objIn.readObject();
            System.out.println(ob2);

// Close the stream
            objIn.close();

        } catch (Exception writeErr) {
// Dump any errors
            writeErr.printStackTrace();
        }
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果用户没有最新版本的 Java,那么用户可以从 www.javasoft.com 上下载对象序列化扩展。

27.12　其它流

Java 还提供了一些实用的过滤器。这些过滤器都不是成对出现的,既它们或者没有输入,或者没有输出。

27.12.1　LineNumberReader 类

LineNumberReader 允许用户输入流的当前行号。通常,用户可以通过向构造器传递一个用户想要过滤的输入流来创建 LineNumberReader:

```
public LineNumberReader(new InputStreamReader(InputStream inStream))
```

getLineNumber 方法返回输入流中当前的行号:

```
public int getLineNumber( )
```

缺省情况下,行号是从 0 开始数起的。当读完一行完整的数据时行号增加一个单位。用户可使用 setLineNumber 方法设置当前的行号:

```
public void setLineNumber(int newLineNumber)
```

清单 27.15 显示了打印带有当前行号的标准输入内容的程序。



```
import java.io.*;

// This class reads lines from standard input (System.in) and
// prints each line along with its line number.

public class PrintLines extends Object {
  public static void main(String[] args) {

    // Set up a line number input filter to count the line numbers
    LineNumberReader lineCounter = new LineNumberReader(new InputStreamReader(System.in));
    try {
      while (true) {

        // Read in the next line
        String nextLine = lineCounter.readLine();

        // If readLine returns null, we're hit the end of the file
        if (nextLine == null) break;

        // Print out the current line number followed by the line
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            System.out.print(lineCounter.getLineNumber());
            System.out.print( * : * );
            System.out.println(nextLine);
          }
        catch (Exception done) {
          done.printStackTrace();
        }
      }
    }
```

27.12.2 SequenceInputStream 类

SequeuceInputStream 过滤器允许用户将多个输入流当作一个大输入流来看待。如果用户希望从多个文件中读取数据但是又不关心哪个文件停下了和哪个文件又开始了，那么这个过滤器就显得非常有用了。如果用户可以很容易地将所有输入文件合并成一个大文件，那么这个流也可以派上用场。用户可以通过向构造器传递两个流来创建一个把这两个流结合在一起的 SequenceInputStream 对象：

```
public SequeuceInputStream(InputStream stream1,InputStream stream2)
```

如果用户希望合并更多的流，那么用户可以向构造器传递一个枚举数：

```
public SequenceInputStream(Enumeration e)
```

该枚举数代表用户希望合并的输入流对象。实现这种做法的一种简单办法是将用户的所有的输入流联结成一个向量，并使用向量的 element 方法调用它们：

```
Vector V = new Vector( );
V.ADDeLEMENT(Stream1);
V.addElement(Stream2);
V.addElement(Stream3);
V.addElement(Stream4);
InpuStream seq = new SequenceInputStream(V.elements( ));
```

如果用户希望合并三个流，那么用户也可以这样创建 SequenceInputStream 链：

```
InputStream seq = new SequenceInputStream(stream1, new SequenceInputStream(stream2, stream3)
```

27.12.3 PushbackInputStream 类

PushbackInputStream 是一个特殊的流，它允许用户监视输入流中的某个字符并将它压回该流中。这种技术经常用于创建词汇扫描器，词汇扫描器监视某个字符，发现后将它压回流中，然后将该字符作为更大的输入的一部分来读取。例如用户可能看到下一个字符是一个数字，这时用户可以将它先放回去，然后调用另一段程序去读取它。现在用户可能已经想象出只要将用户想要过滤的输入流传递给构造器就可以创建 PushbackInputStream 过滤器了。用户的这种想象是完全正确的：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

public PushbackInputStream(InputStream inStream)

unread方法将某个字符压回输入流中：

public void unread(int ch) throws IOException

这个字符是下一次在该流中读取的第一次字符。压回去的字符不必是最近读取的字符。换句话说，用户可以从流中读出一个字符，然后再用unread方法压回另一个字符。用户每次只能使用Unread方法压回一个字符。

27.13 StreamTokenizer类

StreamTokenizer类实现了一个简单的词汇扫描器。它将字符流分解成一个语言符号流。如果用户认为某个字符流是一个句子，那么语言符号就描述了构成该句子的词和发音标志。用户可以通过向构造器传递用户希望过滤的输入流来创建一个StreamTokenizer过滤器：

public StreamTokenizer(Input inStream)

当用户创建了过滤器以后，用户就可以使用nextToken方法从流中获取语言符号：

public int nextToken( ) throws IOException

nextToken方法或者返回一个字符，或者返回下述常量之一：

- StreamTokenizer.TT_WORD
- StreamTokenizer.TT_NUMBER
- StreamTokenizer.TT_EOL
- StreamTokenizer.TT_EOF

如果语言符号返回值为TT_WORD，那么sval实例变量包含的实际是一个单词：

public String sval

如果语言符号值为TT_NUMBER，那么nval实例变量包含的是语言符号的数字值：

public double nval

TT_EOL和TT_EOF分别代表行尾和文件尾。

用户可以使用以字符范围的开始字符和结尾字符作为参数为wordchars方法指定构成某个单词的字符：

public void wordChars(int lowChar, int highchar)

wordChars调用是可增加的，所以后续的wordChar调用可以增加可能的单词字符，记住不是替代而是增加。缺省的单词字符集由下列调用定义：

```
tokenizer.wordChars('A','Z');//全部大写字符
tokenizer.wordChars('A','Z');//全部小写字符
tokenizer.wordChars(150,255);//7位ASCII字符以外的字符
```

如果用户要编写从语法上分析Java程序的程序，那么用户还可能希望在有效单词字符中增加'$'和'_'，因为这两个字符可能出现在Java标识符中，用户可以使用下述两个调用将它们加入有效字符集中：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
toKenize.wordChars('$','$');
toKenize.wordChars('_','_');
```

空白是一个为语言符号定界的字符。空白本身不是语言符号,但是它可以定义一个语言符号的结束和另一个语言符号的开始。例如,短语"Nyuk nyuk nyuk"包含有三个 TT_WORD,它们是被空白隔开的。短语"Nyuk,nyuk,nyuk"实际上包含五个语言符号:三个 nyuk 和两个','读言符号。在这种情况下,空白被忽略了。典型的空白字符如下所示:

- ' '
- '\t'(制表符)
- '\n'(换行符)
- '\r'(回车符)
- '\f'(表格换行符)
- 文件结尾

用户可以使用 whitespaceChars 方法定义空白字符,像 wordChar 方法一样,whitespaceChar 方法也是可以增加的:

```
public void whitespaceChars(int lowChar, int highchar)
```

用户可以使用下述方法定义空白字符集:

```
tokenizer.whitespaceChars(' ',' ');
tokenizer.whitespaceChars('\t','\t');
tokenizer.whitespaceChars('\n','\n');
tokenizer.whitespaceChars('\r','\r');
tokenizer.whitespaceChars('\f','\f');
```

Sun 公司使用了捷径并将所有的控制字符定义为空白,包括空格。换句话说,StreamTokenizer 将像 ESC 符和退格符这样的字符都认为是空白。这样做允许 tokenizer 使用下面的调用来设置它的空白字符:

```
tokenizer.whitespaceChars(0,'');
```

StreamTokenizer 还可以处理注释。它对多字符注释处理得不是太好,对 Java 不使用的类似于引号的注释也处理得不好。它可以处理//风格的注释,也可以处理 Java 和 C++ 使用的/*-*/注释,但是它不能处理 Pascal 中的(*-*)注释。为了使//风格的注释能够被分解,应将 true 传给 slashSlashComments 方法:

```
public void slashSlashComments(boolean allowSlashSlash)
```

为了激活对/*-*/注释的支持,应将 true 传递给 slashSlashComments:

```
public void slashSlashComments(boolean allowSlashStar)
```

除了这两个方法以外,用户还可以使用 commentChar 方法将一个单独的字符标记为注释字符:

```
public void CommentChar(int commentChar)
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

注释字符被认为是单行注释，这意味着一旦遇到注释字符，那么本行后面的内容就被全部忽略掉并从下一行重新开始分析。commentChar 方法也是可以增加的，所以用户可以通过多次调用该方法来设置多个注意字符。

警告：缺省情况下 StreamTokenizer 将/设置成注释字符。用户可能希望使用 ordinaryChar 方法将它设置成普通字符。否则，当 tokenizer 在任何时候遇到/都会将它和直到本行结束前的所有字符全部跳过。

用户可以通过调用 ordinaryChar 方法或 ordinaryChars 方法取消对任何字符所做的特殊设置：

```
public void ordinaryChar(int ch)
public void ordinaryChar(int lowchar, int highChar)
```

这两个方法取消某个字符的任何特殊意义。如果用户在将'$'和'_'字符设置成了单词字符后又觉得它们不应该是单词字符，那么用户就可以使用下述语句将它们设成普通字符：

```
tokenizer.ordinaryChar('$');
tokenizer.ordinaryChar('_');
```

StreamTokenizer 也识别引号字符。当 tokenizer 遇到引号字符时，将它接收存放在 sval 中，然后将引号字符作为语言符号返回。用户可以通过调用 quoteChar 方法将一个字符标记成为引号字符：

```
public void quoteChar(int ch)
```

缺省的引号字符是'和"。

正常情况下，为 TT_WORD 返回的单词是按照它们在输入流中出现的形式存放的。如果用户希望语言符号成为大小写不敏感的，换句话说，如果用户希望 FOO、Foo 和 foo 相等，那么用户可以通过向 lowerCaseMode 方法传递 true 来请求 tokenizer 自动地将单词中的字符全部转成小写：

```
public void lowerCaseMode(boolean shiftToLower)
```

parseNumber 方法告诉 tokenizer 接收浮点数：

```
public void parseNumber( )
```

如果没有调用该方法，tokenizer 将把 3.14159 当作三个独立的语言符号："3"、"."和"14159"。StreamTokenizer 构造器自动地调用该方法。唯一一次需要调用它的机会出现在用户调用 resetSyntax 方法时。

resetSyntax 方法完全地清除 tokenizer 表：

```
public void resetSyntax( )
```

警告：由于 streamTokenizer 被设计成要求空格符(' ')必须是普通字符之外的控制符，在用户调用 resetSyntax 方法之后，用户必须将空格设置成空白、单词字符、注释字符或引号字符。否则 tokenizer 将无法从流中读取字符。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 28 章　使用字符串和文本

本章内容：

28.1　字符串简介
28.2　使用 String 类
28.3　使用 StringBuffer 类
28.4　使用 StringToKenizer 类
28.5　字体处理

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

28.1　字符串简介

C 和C ++(是它们启发了 Java 语言设计)的字符串处理可谓声明狼籍的笨蛋。Java 使用了与C ++ 程序员使用的相同方法解决了这个问题:即通过建立 String 类。Java 的 String 类使得用户的程序可以轻松地管理文本字符串,用户可以使用类似于较简单的语言如 Basic 和 Pascal 的语句。Java 还可以轻松地处理字体,字体可以决定文本字符串在屏幕上显示的样式。

那么什么是字符串呢? 简单地说,字符串就是一个或多个文本字符在内存中的连续排列。用户可以把字符串当作字符数组,这个数组有一个从零开始的索引。(也就是说字符串的第一个字符在数组中的索引为 0)。不幸地是,几乎没有哪种计算机语言以这种简单的形式来处理字符串。这是因为程序需要知道字符串在何处结束,并且字符串有不同长度。例如 Pascal 跟踪字符前面的字符串长度,而C ++ 则寻找字符串结尾的 null(一个零)字符。

在 Java 中,表示字符串有两个类:

- String　适用于字符串常量—即一经创建就不再修改的字符串。
- StringBuffer　适用于需要做许多操作的字符串。

> **注意**:使用 String 类,虽然可以做类似于查找、比较和连接字符等操作,但是不能在串中插入字符或修改字符串的长度(除非做字符串连接—字符串连接实际上是创建了一个新的字符串)。

在 String 类和 StringBuffer 类的对象中,Java 创建了一个字符数组,这一点与C ++ 程序中字符串的使用很类似。因为字符数组被隐藏在类里面,所以除了类的方法以外无法访问它。这种数据封装(面向对象编程的关键特性)保证了字符串按照类的规则由方法执行维护和操作。

图 28.1 和图 28.2 显示了这个概念。在图 28.1 中,常规的C ++ 字符串在内存中可以被程序任意地操作,不管操作是否正确和是否产生致命错误。在图 28.2 中字符虽是受类的方法保护的,但是程序只有通过类的方法才能访问字符串。

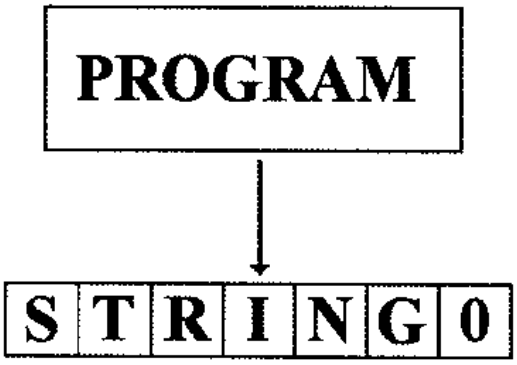


图 28.1　在常规的程序中字符串可以被程序直接访问,所以比较复杂且容易导致错误

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

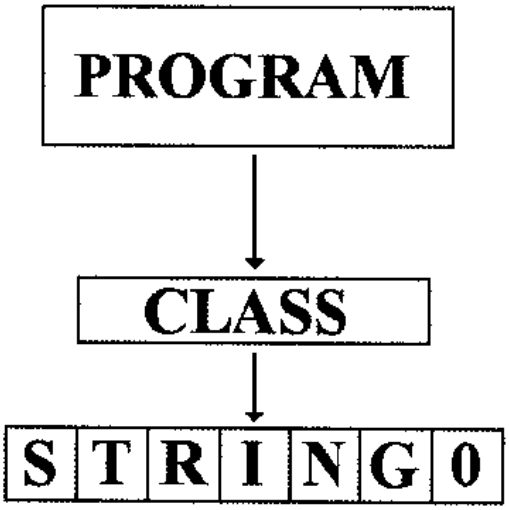


图 28.2 通过使用 String 类,字符串只能被类的方法访问,这样避免了许多潜在的错误

28.2 使用 String 类

在 Java 中,用户可以通过创建 String 类或 StringBuffer 类的对象来创建字符串。String 对象既可以隐式地创建,也可以显式地创建,具体创建形式取决于字符串在程序中的用法。为了隐式地创建一个字符串,用户只要将字符串字符放在程序中,Java 则会自动地创建 String 对象。这是因为在 Java 内部 Java 使用 String 对象表示字符串字符。例如:

```
g.drawString("This is a string",50,50);
```

这时用户正在为字符串"This is a string"隐式地创建 String 对象。每当用户在 Java 程序中如此引用字符串都将创建一个 String 对象。

另一种创建 String 对象的办法是显式地实例化 String 类的一个对象。String 类有七个构造器,所以有丰富的方法可以显式地创建 String 对象,最明显的方法是:

```
String str = new String("This is a String");
```

用户也可以先声明一个 String 对象,然后在以后的程序中为它设置值,例如:

```
String str;
str = "This is a string";
```

或者,用户也可以把这两个语句合并在一起,例如:

```
String str= "This is a string";
```

最后,用户可以使用下面语句创建空字符串:

```
String str = New String( );
String str = "";
```

虽然前面的例子都是最普通的显式地创建字符串对象的方法,但是 String 类提供了几种替代方法。七个 String 类构造器如下所示:

```
public String( )
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public String (String value)
public String (char value[ ])
public String (char value[ ], int offset, int count)
public String (byte ascii[ ],int hibyte, int offset, int count)
public String (byte ascii[ ],int hibyte)
public String (stringBuffer buffer)
```

这些构造器可以分别地创建下列字符串：

- 空字符串
- 来自另一个 String 对象(包含从一个字符串常量)的字符串
- 来自字符数组的字符串
- 来自使用 hibyte 作为每个统一码字节的字节子数组成的统一码字符串
- 来自使用 hibyte 作为每个统一码字节的高字节的字节数组的统一码字符串
- 来自 StringBuffer 对象的字符串

警告：空字符串和空 String 对象之间有很大有区别。当用户使用类似于 String str；这样的语句行声明一个 String 对象时，用户实际上声明了一个未被初始化的一个 String 类的对象。即还没有一个 String 对象与 str 类联，这意味着该 String 对象是空的。当用户使用类似于 String str = "";这样的语句行声明一个 String 对象时，用户实际上声明了一个已经被初始化过了的不包含字符的 String 对象(字符串长数为零)。这种串叫做空字符串。

28.2.1　获取关于 String 对象的信息

当用户构造了 String 对象以后，用户就可以调用 String 类的方法来获得关于该字符串的信息。要想获得字符串的长度，用户可以调用 length( )方法，例如：

```
String str = This is a string;
int len = str.length( );
```

这两行代码将 len 的值设成了 16，即该字符串的长度为 16(当然包括空格)。

如果用户希望知道某个字符串是否是以某个前缀开始，用户就可以调用 startsWith( )方法，例如：

```
string str = This is a string;
boolean result = str.startsWith('This')
```

在这里，布尔变量 result 的值为 true，因为 str 确实是以 This 开头的．在下面例子中 result 的值为 false：

```
String str = "This is a string";
boolean result = str.startsWith('is');
```

类似于 startWith( ) 的方法是 endsWith( )，该方法判断 string 的对象是否以给定的字符集结尾。用户可以像下面这样使用该方法：

```
String str = This is a string;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
boolean result = str.endsWiht('string');
```

在这个例子中,result 的值为 true,而下面的代码段将 result 设置为 false:

```
String str = This is a string;
boolean result = str.endsWith('This');
```

如果用户为了快速定位而为字符串设置了一个表,那么用户可以使用散列表。为了获得某个字符串的散列代码,用户可以调用 hashCode( )方法:

```
String str = This is a string;
Int hashcode = str.hashCode( );
```

参见第 46 章的 Hashtable 类。

如果用户想找到在某个字符串的第一次出现的位置,用户可以使用 indexOf( )方法:

```
String str = This is a string;
Int index = str.indexOf( 'a');
```

在本例中,index 的值等于 8,即在字符串中 A 字符的索引是 8。

为了找到后续的字符的位置,用户就可以使用两个版本的 indexOf( )方法。为了找到第一次出现的 I,用户可以使用下面的两行语句:

```
String str = This is a string;
Int index = str.indexOf( 'i');
```

在这里 index 的结果是 2. 为了找到下一个 'i', 用户可以使用类似于下一行的语句:

```
index = str.indexOf("i",index + 1);
```

通过将 index + 1 作为该方法的第二个参数,用户告诉 Java 从字符中索引处开始查找(老的 index 值加 1)。这时候 index 的结果等于 5,即该字符串中第二个"i"字符出现在索引值为 5 处。如果用户再一次调用前面的语句,index 的值将等于 13,这个索引值是第 3 个"i"字符在该字符串中出现的位置。

用户也可以使用 lastIndexOf( ) 方法查找某个字符在字符串中最后一次出现的位置:

```
String str= "This is a String";
int index = str.lastIndexOf("i");
```

在这里 index 的值等于 13。为了反向查找下一个"i"字符,用户可以使用类似于下一行的语句:

```
index = str.lastIndexOf('i',index - 1);
```

现在 index 的值等于 5,因为作为第二个参数的 index - 1 告诉 Java 从 index - 1 开始反向查找最后一个"i"字符。当第一次调用 lastIndexOf()之后 index 的值为 13,当开始第二次调用 lastIndexOf( )时 index - 1 的值等于 12。

indexOf( )方法和 lastIndexOf( )方法寻找字符串的子串。下述示例将 index 设置为 10:

```
String str = "This is a String";
int index = str.indexOf("String")
```

程序清单 28.1 是一个小应用程序,它给出机会让用户体验 indexOf( )方法。程序清单

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

28.2 是加载该小应用程序的 HTML 文档。当用户运行该小应用程序时，应在第一个文本框内输入一个字符串，而在第二个文本框中输入前面字符串的子串。当用户单击 Search 接钮时，小应用程序显示子字符串开始位置的索引值(见图 28.3)。

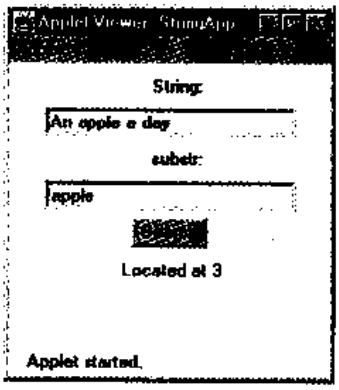


图 28.3　StringApplet 小应用程序查找子字符串

```
import java.awt.*;
import java.applet.*;
public class StringApplet extends Applet
{
    TextField textField1;
    TextField textField2;
    Button button1;
    String displayStr;
    public void init()
    {
        Label label = new Label("String:");
        add(label);
        textField1 = new TextField(20);
        add(textField1);
        label = new Label("Substr:");
        add(label);
        textField2 = new TextField(20);
        add(textField2);
        button1 = new Button("Search");
        add(button1);
        displayStr = "";
        resize(230, 200);
    }
    public void paint(Graphics g)
    {
        g.drawString(displayStr, 80, 150);
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public boolean action(Event evt, Object arg)
    {
        if (arg == "Search")
        {
            String str = textField1.getText();
            String substr = textField2.getText();
            int index = str.indexOf(substr);
            displayStr = "Located at " + str.valueOf(index);
            repaint();
            return true;
        }
        else
            return false;
    }
}
```

```
<title>Applet Test Page</title>
<h1>Applet Test Page</h1>
<applet
    code="StringApplet.class"
    width=200
    height=200
    name=<"tringApplet">
</applet>
```

28.2.2 比较字符串

用户经常需要知道何时两个字符串相等。例如用户可能希望比较输入的字符串和程序中另一个硬编码字符串是否相等。这里有两种比较字符串的办法：

- 调用 equals( )方法
- 使用普通的比较运算符

当两个字符串相等时 epuals( )方法返回 true，否则返回 false。例如：

```
String str = "This is a String";
if (str == "This is a String")
    result = true;
```

这将导致 result 的值为 true。

虽然这两个方法是比较字符串最容易的办法，但是 String 类还是给出了许多其它选项。equalsIgnoreCase( )方法可以以忽略大小写的方式比较两个字符串。下述代码片段将 result 的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

值设为 false,因为 equals( )考虑字符串中字符的大小写差别:

```
String str = "THIS IS A STRING";
boolean result = str.equalsIgnoreCase("this is a string");
```

如果用户希望知道除字符串相等以外的情况,那么用户可以调用 compareTo( )方法,当字符串对象小于给定字符串时 compareTo( )方法返回小于零的值,当字符串相等时返回零,如果字符串对象大于给定字符串时返回大于零的值。比较是根据字母顺序进行的(或者,如果用户严格地讲,比较是根据字符的 ASCII 值进行的)。所以,下述代码片段将 result 的值设为大于零的值,因为"THIS IS A STRING"大于"ANOTHER STRING"

```
String str = "THIS IS A STRING";
int result = str.compareTo("ANOTHER STRING");
```

在下述的比较操作中 result 的值被设置成为小于零的值,因为"THIS IS A STRING"小于"ZZZ ANOTHER STRING":

```
String str = "THIS IS A STRING";
int result = str.compareTo("ZZZ ANOTHER STRING");
```

最后,下述比较的结果为零,因为两个字符串相等:

```
String str = "THIS IS A STRING";
int result = str.compareTo("THIS IS A STRING");
```

C 和C++程序员应该非常熟悉这种字符串比较格式。

如果用户想把字符串比较做得更漂亮,那么使用 regionMathes( )方法会让用户的编程伙伴感到妙不可言,这个方法允许用户将一个字符串的一部分与另一个字符串比较。例如:

```
String str = "THIS IS A STRING";
boolean result = str.regionMatches(10, "A STRING",2,6);
```

regionMatches( )方法的四个参数的含意是:

- 在源字符串中要比较的子串的开始位置
- 要比较的字符串
- 比较字符串中要比较的子串开始位置
- 要比较的字符数

前面的例子将 result 的值设置为 true 。在这里,Java 从"THIS IS A STRING"字符串第 10 个字符开始查找(从 0 开始计数),即从 STRING 中的 S 开始。Java 还从给定字符串"A STRING"的第二个字符开始,即也是从 STRING 中的 S 开始。Java 按照给定的偏移量比较六个字符,这意味着比较的内容是 STRING 和 STRING,它们是完全匹配的,所以 result 的值被设置为 true。

下述代码片段中的 regionMatches( )方法的版本是忽略字符大小写的。该示例也将 result 设置为 true:

```
String str = "THIS IS A STRING";
boolean result = str.regionMatches(true,10, "A String",2,6);
```

新版本的 regionMatches( )方法的第一个参数是一个布尔值,它指示比较是否区分大小

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

写。true 值告诉 Java 忽略字符的大小写。该参数的 false 值告诉用户执行与带四个参数的 regionMatches( )版本相同的比较。

程序清单 28.3 是一个小应用程序，该小应用程序给用户一个机会去体验 compareTo( )方法。程序清单 28.4 是运行该小应用程序的 HTML 文档。当用户运行该小应用程序时，用户需要在每个文本框中输入一个字符串。当用户单击 Compare 按钮时，小应用程序确定字符串比较结果并将结果显示出来(如图 28.4 所示)

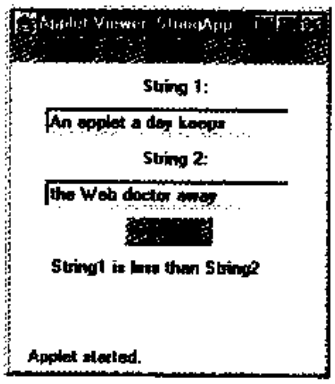


图 28.4 StringApplet2 比较两个字符串

```
import java.awt.*;
import java.applet.*;
public class StringApplet2 extends Applet
{
    TextField textField1;
    TextField textField2;
    Button button1;
    String displayStr;
    public void init()
    {
        Label label = new Label("String 1:");
        add(label);
        textField1 = new TextField(20);
        add(textField1);
        label = new Label("String 2:");
        add(label);
        textField2 = new TextField(20);
        add(textField2);
        button1 = new Button("Compare");
        add(button1);
        displayStr = "";
        resize(230, 200);
    }
    public void paint(Graphics g)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        {
            g.drawString(displayStr, 30, 150);
        }
        public boolean action(Event evt, Object arg)
        {
            if (arg == "Compare")
            {
                String str1 = textField1.getText();
                String str2 = textField2.getText();
                int result = str1.compareTo(str2);
                if (result < 0)
                    displayStr = "String1 is less than String2";
                else if (result == 0)
                    displayStr = "String1 is equal to String2";
                else
                    displayStr = "String1 is greater than String2";
                repaint();
                return true;
            }
            else
                return false;
        }
    }
```

```
<title>Applet Test Page</title>
<h1>Applet Test Page</h1>
<applet
    code="StringApplet2.class"
    width=200
    height=200
    name="StringApplet2">
</applet>
```

28.2.3　字符串抽取

在用户的编程经历中可能多次想从一个字符串中抽取一部分。String 类为这种需要提供了一套方法。例如用户可以通过调用 charAt( )方法确定字符串中给定位置上的字符：

```
String str= "This is a string";
Char chr =str.charAt(6);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这两行语句将字符变量 chr 的值设置为“s”，它是该字符串的第六个字符。为什么 chr 不等于 i 呢？因为像 C 和C++一样，数组元素是从零开始计数的。

类似的方法，getChars( )允许用户将 String 对象的一部分拷贝到字符数组中：

```
String str = "This is a String";
char chr[ ] = new char[20];
str.getChars(5,12,chr,0);
```

在这段代码示例中，字符串数组 chr 最终包含的字符是“is a st”。getChars( )方法的参数是要拷贝的字符串中第一个字符串的索引、该串中最后一个字符的索引、目标数组和在目标数组中开始拷贝字符串的位置。

getBytes( )方法做与 getChars( )方法相同的事情，但是它使用字节数组作为目标数组：

```
String str = "This is a string";
byte tyt[ ] = new byte[20];
str.getBytes(5,12,byt,0);
```

抽取部分字符串的另一种方式是使用 substring( )方法：

```
String str1 = "THIS IS A STRING";
String str2 = str1.substring(5);
```

在这里 String 对象 str2 的结果为子字符串“IS A STRING”。这是因为 substring( )方法的唯一参数是子字符串开始字符的索引。从该索引到该字符串结尾之间的每个字符全部被抽取出来。

如果用户不希望抽取到字符串结尾之前的所有字符，那么用户可以使用 substring( )方法的第二个版本，它拥有两个参数分别指定字符串的开始和结尾：

```
String str1 = "THIS IS A STRING";
String str2 = str1.substring(5,9);
```

这两行语句将 str2 设置成子字符串“IS A”。

程序清单 28.5 是一个小应用程序，它给用户一个体验 indexOf( )方法的机会。程序清单 28.6 是运行该小应用程序的 HTML 文档。当用户运行该小应用程序时，用户需要首先在第一个文本框中输入一个字符串。然后，在第二个框和第三个框中输入子字符串的开始索引和结束索引。当用户单击 Extract 按钮时，小应用程序找出并显示被选中的子字符串（如图 28.5 所示）。

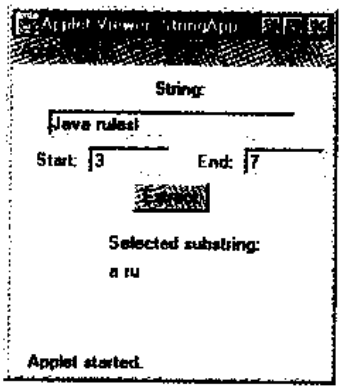


图 28.5 在小应用程序浏览器中运行 StringApplet3

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

警告：在程序清单 28.5 中没有错误检查，所以用户需要确保索引值是正确的。否则 Java 将产生异常。

程序清单 28.5　StringApplet3.java——[illegible]

```
import java.awt.*;
import java.applet.*;
public class StringApplet3 extends Applet
{
    TextField textField1;
    TextField textField2;
    TextField textField3;
    Button button1;
    String displayStr;
    public void init()
    {
        Label label = new Label("String:");
        add(label);
        textField1 = new TextField(20);
        add(textField1);
        label = new Label("Start:");
        add(label);
        textField2 = new TextField(5);
        add(textField2);
        label = new Label("End:");
        add(label);
        textField3 = new TextField(5);
        add(textField3);
        button1 = new Button("Extract");
        add(button1);
        displayStr = "";
        resize(230, 200);
    }
    public void paint(Graphics g)
    {
        g.drawString("Selected substring:", 70, 130);
        g.drawString(displayStr, 70, 150);
    }
    public boolean action(Event evt, Object arg)
    {
        if (arg == "Extract")
        {
            String str1 = textField1.getText();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            String str2 = textField2.getText();
            String str3 = textField3.getText();
            int start = Integer.parseInt(str2);
            int end = Integer.parseInt(str3);
            displayStr = str1.substring(start, end);
            repaint();
            return true;
        }
        else
            return false;
    }
}
```

```
<title>Applet Test Page</title>
<h1>Applet Test Page</h1>
<applet
    code="StringApplet3.class"
    width=200
    height=200
    neme="StringApplet3"
</applet>
```

28.2.4 字符串操作

虽然 String 类是用于处理字符串常量的,但是该类提供了一些字符串操作的方法用以"修改"String 对象。这里的修改一词是用引号引起来的,因为这些字符串操作方法并不实际地改变 String 对象,它创建了一个额外的 String 对象,用来具体实现要求的修改。其中 replace( ) 方法允许用户用另外一个字符代替字符串中的任何字符:

```
String str1 = "THIS IS A STRING";
String str2 = str1.replace('t', 'x');
```

在这个示例中,str2 包含的字符串为"XHIS IS A STXRING",因 replace( )方法调用要求用 X 代替原字符串中的每一个 T 字符,注意 str1 保持不变,str2 是一个新的字符串;

用户可以用于操作字符串的另一个办法是联接两个字符串。当用户联接两个字符串时,用户得到一个包含原来两个字符串的新字符串。例如:

```
String str1 = "THIS IS A STRING";
String str2 = str1.concat("XXXXXX");
```

这里,str2 包含的字符是"THIS IS A STRINGXXXXX X",而 str1 保持不变。正如用户可以看到的那样,concat( )方法只有一个参数是要被联接到原字符串上的字符串。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

为了简化操作，String 类定义了一个运算符，即加号(+)，它执行的操作就是联接两个字符串。通过使用加号运算符用户可以更直观地联接字符串。下面是一个示例：

```
String str1 = "THIS IS A STRING";
String str2 = str1 + "XXXXXX";
```

这个代码片段产生与前面的 concat( )示例完全一样的字符串。这意谓用户可以在同一行中多次使用联接运算符，例如：

```
String str = "This" + "is" + "a test";
```

如果用户希望获得全部大写或全部小写的字符串，那么用户可以使用 toUpperCase( )方法和 toLowerCase( )方法，这两个方法返回的字符串里的字符被全部转换为大写或小写。例如，看一下下述两行：

```
String str1 = "THIS IS A STRING";
String str2 = str1.toLowerCase( );
```

在这里，str2 为“this is a string”，因为 toLowerCase( )方法将字符串中的所有字符串转换为小写。当然 toUpperCaese 方法恰恰相反，它将所有字符转换为大写。

有时字符串前后有空格，String 类的 trim( )方法能够删除这种空白字符。例如：

```
String str1 = "          THIS IS A STRING"
String str2 = str1.trim( );
```

在本例中 str2 包含的字符串为“THIS IS A STRING”，第一个 T 字符前面的空格和最后一个 G 后面的空格全部被删除掉。

最后，用户可以使用 String 类的 valueOf( )方法将任意数据对象转换成字符串，这样可以在屏幕上显示对象值。下面两行语句将整数转换成字符串：

```
int value = 10;
String str = String.valueOf(value);
```

注意 valueOf( )是一个静态方法，用户可以通过引用 String 类直接调用它，不必实例化 String 对象。当然，用户也可以通过 String 类的任何对象调用 valueOf( )方法，例如：

```
int value = 10;
String str1 = "";
String str2 = str1.valueOf(value);
```

28.3　使用 StringBuffer 类

StringBuffer 类允许用户创建可以以各种方式修改的 String 对象，它与描述字符串常量的 String 类不同。当用户使用 StringBuffer 类的字符串时，用户并没有创建一个新的 String 对象，而是直接操作原字符串本身。为此，StringBuffer 类提供了与 String 类不同的方法集，所有的操作都直接发生在包含该字符串的缓冲区上。

28.3.1　创建 StringBuffer 对象

StringBuffer 类提供的几个构造器允许用户以各种方式构造 StringBuffer 对象。这些构造

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

器是：

```
StringBuffer( )
StringBuffer(int length)
StringBuffer(String str)
```

这三个构造器分别创建空的 StringBuffer 对象、长度为 length 的空 StringBuffer 对象和用 String 对象(或字符串常量)创建的 StringBuffer 对象。

28.3.2 获取关于 StringBuffer 对象的信息

就像使用规则的字符串那样，用户可能需要知道存放在 StringBuffer 对象中字符串的长度。该类为此提供了 length( )方法。StringBuffer 对象还提供了 capacity( )方法，该方法返回缓冲区的容量。StringBuffer 的长度是存放在字符串中字符的数量，而容量是缓冲区可容纳字符的最大数量。在下面的示例中，长度是 2，容量 28：

```
StringBuffer str = new StringBuffer("xx");
int length = str.length( );
int capacity = str.capacity( );
```

28.3.3 StringBuffer 对象

在学习 String 类时用户已经体验到了字符串抽取。StringBuffer 类提供了与 String 类完全相同的两个方法来完成这项任务。这两个方法是 charAt( )和 getChars( )，这两方法的用法与 String 版本的这两个方法类似。下面是使用 charAt( )方法的示例：

```
StringBuffer str = new StringBuffer("String buffer");
char ch = str.charAt(5);
```

下面是使用 getChars( )方法的示例：

```
StringBuffer str = new StringBuffer("String buffer");
char ch[ ] = new char[20];
str.getChars(7,10,ch,0);
```

另外，在使用 JDK1.2 时用户还可以从 StringBuffer 类中得到 subString( )方法，就像 String 类一样。下面是一个示例：

```
StringBuffer str = new StringBuffer("String buffer");
String buffer = str.subString(8);
String rin = str.subString(3,6);
```

28.3.4 操作 StringBuffer 对象

存放在 StringBuffer 对象中的字符串可以有多种修改办法。和 String 类中的字符串修改方法不同，StringBuffer 类不必创建新字符串，可以直接在存放原字符串的缓冲区上工作。用户在字符串缓冲区上可以做的第一件事情是设置它的长度。用户可以通过调用 setLength( )方法来设置字符串长度：

```
StringBuffer str = new StringBuffer("String buffer");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
str.setLength(40);
```

这个方法唯一的一个参数是新的长度。如果新长度大于老长度，那么字符串长度和缓冲区长度都被增加，额外的字符使用零来填充。如果新长度小于老长度，那么结尾后面的字符被砍掉，但是缓冲区尺寸保持不变。

如果用户希望保证一个特定的缓冲区尺寸，那么用户可以调用 ensureCapacity( )方法，像这样：

```
StringBuffer str = new StringBuffer("String buffer");
str.ensureCapacity(512);
```

ensureCapacity( )方法的参数是该缓冲区新的容量。

用户可以调用 setCharAt( )方法来修改字符串缓冲区中的某个字符：

```
StringBuffer str = new StringBuffer("String buffer");
str.setCharAt(3, 'X');
```

setCharAt( )方法的两个参数是要修改字符的索引和要修改成的新字符。在前面的示例中，字符串缓冲区变成“StrXng buffer”。在 JDK1.2 中用户也可以通过 replace( )方法使用 String 或者它本地的类型替换 String 的一部分。replace( )方法类似于 setCharAt( )方法，还过它不仅仅可以替换一个字符，而且可以替换其它字符串类；如下一个例子所示。注意在使用 insert( )时用户必须提供被替换字符的开如索引(包含在内)和结束索引(不包含在内)，而 setCharAt( )仅仅提供一个字符索引就可以。

```
StringBuffer str = new StringBuffer("String buffer");str. replace(3,6 'XYZ');
```

用户可以使用 append( )方法在字符串结尾增加字符，也可以使用 insert( )方法在字符串任意位置插入字符。这两个方法都有几个版本使得用户可以处理许多不同类型的数据。为了在字符串末尾增加整数的字符版本，用户应该按照下述方式处理：

```
StringBuffer str = new StringBuffer("String buffer");
int value = 15;
str.insert(6,value);
```

这个代码片段产生的结果是“String buffer15”。在前面的 insert( )版本中的两个参数是插入字符的位置索引和要插入的数据对象。

在 JDK1.2 中用户也可以使用 deleteCharAt( )和 delete( )方法从 StringBuffer 中删除字符。deleteCharAt( )方法要求要删除的字符的索引，delete( )要求开始索引(包括)和结束索引(不包括)，如下面的例子所示：

```
StringBuffer str = new StringBuffer("String buffer");
str.deleteCharAt(6);
str.delete(3,6);
```

28.4　使用 StringToKenizer 类

如果用户使用计算机有一定的历史的话，那么用户可能还记得老式冒险游戏的文本界面。在这种游戏中，当用户从键盘上输入一条类似于 GET KEY AND OPED DOOR 的命令时，计

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

算机将随着用户的指令运行。像这样的程序必须分析字符串并将字符串分解成独立的单词。这些单词叫做语言符号,当从文本串中抽取语言符号时要运行多次。Java 为了这种目的提供了 StringToKenizer 类。

因为 StringToKenizer 类不像 String 类和 StringBuffer 类那样是 java.lang 包的一部分,用户必须在小应用程序中包括正确的包。这个包是 java.util 包,用户应该使用下述语句将它引入:

```
import java.util. StringToKenizer;
```

如果用户想引入整个 util 包,那么用户可以写成这个样子:

```
import java.util. * ;
```

用户可以用多种方式构造 StringToKenizer,其中最容易的方式是作为唯一的参数向构造器提供想要分析字符串,例如:

```
StringToKenizer tokenizer = new StringToKenizer("One Two Three Four Five");
```

这种字符串分析使用空格符作为语言符号之间的分隔符(或定界符)。要想获得一个语言符号,用户可以调用 nextToken( )方法:

```
String token = tokenizer.nextToken( );
```

每当用户调用 nextToken( )时,都将在字符串中获得下一个语言符号。通常,用户可以使用 while 循环来抽取语言符号。为了控制 while 循环,用户可以调用 hasMoreTokens( )方法,只要字符串中还有更多的语言符号,hasMoreTokens( )方法就返回 True。典型的语言符号分析循环如下所示。

```
while(tokenizer.hasMoreTokens( ))
        String token = tokenizer.nextToken( );
```

用户也可以通过调用 countTokens( )方法测定字符串中有多少语言符号:

```
StringTokenizer tokenizer =
new StringTokenizer("One Two Three Four Five");
int count = tokenizer.countTokens( );
```

在本例中 count 等于 5。

程序清单 28.7 是一个小应用程序,该小应用程序分析用户输入的任何字符串。当用户运行该小应用程序时,用户需要在第一个文本框中输入一个字符串。然后单击 Tokenize 按钮就可以得到字符串中的语言符号列表(如图 28.6 所示)。

```
import java.awt. * ;
import java.applet. * ;
import java.util.StringTokenizer;
public class TokenApplet extends Applet
{
    TextField textField1;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    Button button1;
    public void init()
    {
        textField1 = new TextField(30);
        add(textField1);
        button1 = new Button("Tokenize");
        add(button1);
        resize(300, 300);
}
    public void paint(Graphics g)
    {
        String str = textField1.getText();
        StringTokenizer tokenizer =
            new StringTokenizer(str);
        int row = 110;
        while (tokenizer.hasMoreTokens())
        {
            String token = tokenizer.nextToken();
            g.drawString(token, 80, row);
            row += 20;
        }
    }
    public boolean action(Event evt, Object arg)
    {
        if (arg == "Tokenize")
        {
            repaint();
            return true;
        }
        else
            return false;
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

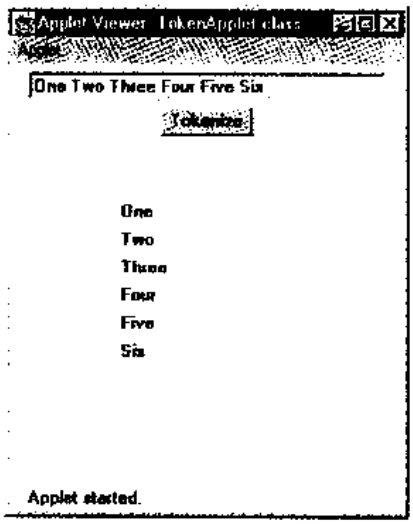


图 28.6 TokenApplet 可以从字符串中抽取独立的单词

28.5 字体处理

因为每个系统都以不同的方式处理字体，所以用户必须在用户的小应用程序中注意使用字体。虽然 Java 的字体匹配做得很好，但是用户还是必须仔细地处理字体以确保显示结果正确。为了帮助用户完成这项任务，Java 提供了 Font 类，Font 类允许用户不仅可以创建和显示字体，而且还可以获得关于字体的信息。因为在 Java 程序中所有文本显示都使用当前字体（包括在像按钮这样的组件上），所以与字符串有关的章节如果没有对字体的讨论都是不完整的。

28.5.1 获取字体属性

用户在自己的 Java 小应用程序中使用的每种字体都与一组决定外观和大小的属性相关联。这些属性中最重要的是字体名，字体名决定了字体的基本风格。用户可以很容易地获得关于当前活动的字体的信息。下面是 Graphics 对象的 getFont( )方法：

```
Font font = g.getFont( );
```

getFont( )方法返回描述当前字体的 Font 对象。当用户拥有了 Font 对象之后，用户就可以使用 Font 类的各种方法去获取关于字体的信息。表 28.1 显示了 Font 类最普通的公共方法和它们的用途。

表 28.1 Font 类中最常用的公共方法

方　法	描　述
GetFamily( )	返回字体的家族名
getName( )	返回字体名
getSize( )	返回字体的尺寸
getStyle( )	返回字体的风格，0 为正常体，1 为粗体，2 为斜体，3 为粗斜体
isBold( )	返回指示字体是否为黑体的布尔值
isItalic( )	返回指示字体是否为斜体的布尔值
isPlain( )	返回指示字体是否为正常体的布尔值
toString( )	返回关于字体的信息字符串

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意：最普通的字体处理方法也可以在小应用程序类中获得。例如，用户可以在小应用程序的 init( )方法中调用 getFont( )方法，不必关心 Graphics 对象。对于 getFontMetrics( )方法和 setFont( )方法也是这样。getFontMetrics 方法和 setFont( )方法将在本章的“获得字体度量”和“使用字体”部分中介绍。

和往常一样，看某个东西如何工作的最好办法就是试用它一下。程序清单 28.8 是一个小应用程序，该小应用程序使用表 28.1 中描述的多个方法显示关于当前字体的信息。图 28.7 显示了在小应用程序浏览器中运行的该小应用程序。

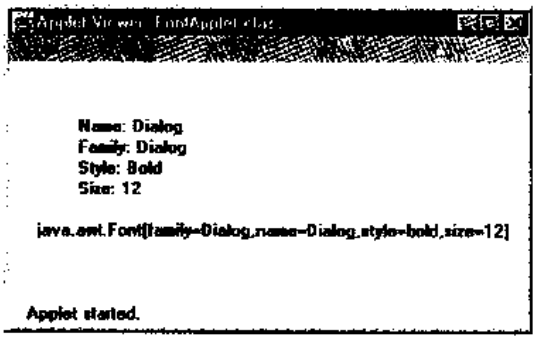


图 28.7　运行在小应用程序浏览器中的 FontApplet

```
import java.awt.*;
import java.applet.*;
public class FontApplet extends Applet
{
    public void paint(Graphics g)
    {
        Font font = getFont();
        String name = font.getName();
        String family = font.getFamily();
        int n = font.getStyle();
        String style;
        if (n == 0)
            style = "Plain";
        else if (n == 1)
            style = "Bold";
        else if (n == 2)
            style = "Italic";
        else
            style = "Bold Italic";
        n = font.getSize();
        String size = String.valueOf(n);
        String info = font.toString();
        String s = "Name: " + name;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            g.drawString(s, 50, 50);
            s = "Family: " + family;
            g.drawString(s, 50, 65);
            s = "Style: " + style;
            g.drawString(s, 50, 80);
            s = "Size: " + size;
            g.drawString(s, 50, 95);
            g.drawString(info, 20, 125);
        }
    }
```

正如用户在程序清单 28.8 中看到的那样，使用 Font 类的方法是很简单的事情。只要调用这些方法就可以返回由 Font 对象描述的关于字体的某些值。

28.5.2 获取字体度量

在大部分情况下用户从 Font 对象中获得的信息就足够用了。例如通过调用 getSize( ) 方法，用户可以正确地留出文本行的空间。但是有些时候用户可能希望知道更多的关于正在使用的字体的信息。例如用户可能希望知道特定字符的宽度，甚至整个字符串的像素宽度。在这些情况下，用户需要使用文本度量，文本度量更详细地描述了字体属性。

Java 开发工具一如既往地为当前字体包括了 FontMetrics 对象。它可以很容易地获取关于字体的信息。用户可以像下面这样创建 FontMetrics 对象：

```
FontMetrics fontMetrics = getFontMetrics(font);
```

getFontMetrics( )方法返回对当前字体的 FontMetrics 对象的引用。它们唯一一个参数是用户希望了解其字体度量的 Font 对象。

当用户拥有了 FontMetrics 方法以后，就可以调用它的方法来获取与该字体关联的详细信息。表 28.2 列出了其中最常用的方法。

表 28.2 常用的 FontMetrics 方法

方　法	描　述
CharWidth( )	返回字符的宽度
GetAscent( )	返回字体的顶线
GetDescent( )	返回字体的底线
GetFont( )	返回关联的 Font 对象
getHeight( )	返回字体的高度
getLeading( )	返回字体行距(行间空隙)
StringWidth( )	返回字符串的宽度
toString( )	返回一个包含字体信息的字符串

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

提示：如果用户以前没有使用字体，可能对某些字体术语并不熟悉，比如表 28.2 中的顶线、底线等。行距是文本行之间的空白量。顶线是由基线算起的高度，即从字符的基线到字符顶的距离。底线是字符在基线下面伸出来部分的高度，比如小写 g。字符高度是顶线、底线和行距三者之和，如图 28.8 所示。



图 28.8　顶线、底线和行距决定了字体的高度

28.5.3　创建字体

用户可能认为总是使用一种字体会腻烦的。在许多情况下这种观点是正确的。把小应用程序装扮整齐的简单办法是使用不同字体。幸运的是，Java 允许用户为小应用程序创建和设置字体。用户可以创建自己的字体，例如：

```
Font font = new Font("TimeRoman",Font.PLAN,20);
```

Font 类的构造器使用三个参数：字体名、风格和尺寸。风格可以是定义在 Font 类中的字体属性的任意组合。这些属性是 Font.PLAN、Font.BOLD 和 Font.ITALIC。虽然用户可以使用正常体、粗体和斜体风格创建字体，但是有时用户可能需要组合使用字体风格。例如，假设用户希望同时使用粗体和斜体风格。那么就可以使用类似于下述形式的语句：

```
Font font = new Font("Courier",Font .BOLD + Font.ITALIC,18);
```

该语句给出了 18 点高粗斜体风格的字体。

注意：一个点的字体高度度量相当于 1/72 英寸。

28.5.4　使用字体

当创建完字体以后，用户必须告诉 Java 使用字体。用户可以通过调用 setFont( )方法来实现，例如：

```
setFont(font);
```

这样在用户的小应用程序中接下来显示的文本将使用新的字体。虽然用户要求使用一定的字体类型的尺寸，但是未必能够保证用户将来能否得到实际的效果。系统努力匹配字体，但是用户仍然需要知道最终字体的尺寸。用户可以通过创建 FontMetrics 对象来获得用户需要的全部信息，例如：

```
FontMetrics fontMetrics = getFontMetrics(font);
```

为了获得文本行高度，可以调用 FontMetrics 对象的 getHeight( )方法，例如：

```
int height = fontMetrics.getHeight( );
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

警告:当用户创建某种字体时,系统中可能并没有加载这种字体。在这种情况下,Java使用缺省字体代替被选中的字体。这时用户可以使用 Font.getName( )方法查看用户是否得到了希望使用的字体。用户特别需要知道字体的尺寸,以确保文本行出现在适当的位置。

如果用户不想显示文本是不会创建字体的。问题是在用户可以显示文本之前,用户需要至少知道字体的高度。不考虑字体的高度可能会出现文本覆盖和间距太大现象。用户可以使用 FontMetrics 类的 getHeight( )方法返回的高度作为打印文本的行增量。程序清单 28.9 是 FontApplet2 小应用程序的源代码,该小应用程序显示了如何设置行增量。图 28.9 显示了该小应用程序的运行结果。

程序清单 28.9 FontApplet2.java——显示字体尺寸的小应用程序

```
import java.awt.*;
import java.applet.*;
public class FontApplet2 extends Applet
{
    TextField textField;
    public void init()
    {
        textField = new TextField(10);
        add(textField);
        textField.setText("32");
    }
    public void paint(Graphics g)
    {
        String s = textField.getText();
        int height = Integer.parseInt(s);
        Font font = new Font("TimesRoman", Font.PLAIN, height);
        g.setFont(font);
        FontMetrics fontMetrics = g.getFontMetrics(font);
        height = fontMetrics.getHeight();
        int row = 80;
        g.drawString("This is the first line.", 70, row);
        row += height;
        g.drawString("This is the second line.", 70, row);
        row += height;
        g.drawString("This is the third line.", 70, row);
        row += height;
        g.drawString("This is the fourth line.", 70, row);
    }
    public boolean action(Event event, Object arg)
    {
        repaint();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        return true;
    }
}
```

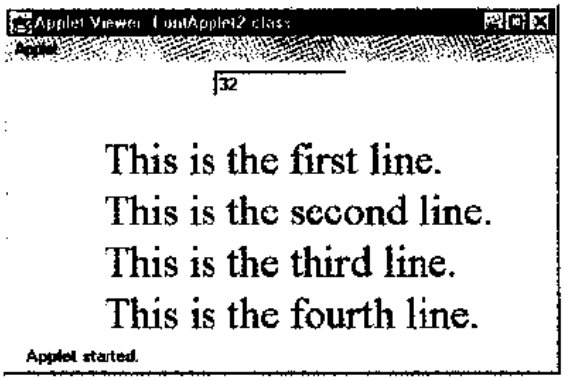


图 28.9　在小应用程序浏览器中运行的 FontApplet2

当用户运行 FontApplet2 时，用户可以看到如图 28.9 所示的窗口。当前字体的尺寸显示在小应用程序顶部的文本框中，文本框下方显示的是该字体的示例。为了改变字体的尺寸，可以在文本框内输入新的值并按下 Enter 键。

行间空是由 row 变量设置的，该变量说明了下一行文本的垂直位置：

```
int row = 80;
```

这里，程序不仅声明了 row 变量，而且还用第一行文本的垂直位置初始化了该值。

接着，小应用程序打印了第一行文本，在这里 row 作为第三个参数出现在 drawString( ) 方法中：

```
g.drawString("This is the first line",70,row);
```

在准备打印下一行文本时，该程序在 row 变量中加入了一个字体高度增量：

```
row + = height;
```

每一行打印文本，都使用字体高度作为 row 的增量，像这样：

```
g.drawString ("This is the second line",70,row);
row + = height;
g.drawString("This is the third line",70,row);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 29 章　使用多国语言实现

本章内容：

29.1　为什么使用多国语言实现
29.2　什么是多国语言实现
29.3　Java 对多国语言实现的支持
29.4　多国语言实现的输入输出(I/O)
29.5　新的 Java.text 包
29.6　示例：InternationalTest

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

29.1　为什么使用多国语言实现

Joe Programmer 是 X 公司的 Java 开发人员。他开发的应用程序在他公司所在地加利福尼亚获得了销售方面的巨大成功，这主要因为他使用了好的面向对象设计和实现；使得他的代码具有可移植性、可重用性和独立性。一天，X 公司决定在日本销售它们产品，Joe Programmer 不懂日语，他找了一位懂 Java 的日语翻译浏览了他的程序并使用某个 Joe 不懂的自定义日本语言字符集对程序做了全面的修改。他很幸运地编译了他的代码的日语版本并将它发往日本，在日本该程序也获得了很大的成功。受这一结果的鼓舞，X 公司开始进入其它市场，接下来是法国和加拿大。令 Joe 沮丧的是，他发现他必须同时维护几个完全不同的版本，因为法国和加拿大虽然使用同一种语言，但却有完全不同的文化！可怜的 Joe 有五个版本的编译代码：美国英语、日语、法语、加拿大法语和加拿大英语。现在，只要他对代码哪怕是只做一个最轻微的改动，就必须做五次同样的改动，然后再雇五个翻译直接在源代码中做语言改动。很显然，Joe Programmer 处于一个难以接受的境地。

29.2　什么是多国语言实现

在前面的叙述中，可以说 Joe Programmer 编写了一个“缺乏远见”的程序，它只适合于一个场所。这里讲的场所（通常指的是地理上的，但也不是必须的）是指这样一个概念：有着共同的习惯、文化和语言的用户群体。Joe 的程序的五个版本都是针对特定地区开发的，未经必要的改动是无法适用于其它地区的。这种结果违背了面向对象设计的基本原则，因为 Joe 的程序不再是可移植和可重用的。这种将依赖于文化的代码（文本、图片等）与和语言无关的代码分离开的处理就叫做多国语言实现。当程序经过了这种处理后就可以很容易地适用于任何场所，当然仍然需要做数量非常小的改动。Java 语言的 1.1 版本增加了内置的多国语言实现支持，它可以使得开发者轻而易举地开发可移植的代码。

29.3　Java 对多国语言实现的支持

多国语言实现要求对 Java 语言做几项修改。在过去，编写多国语言实现的代码要比编写适用于特定地区的代码付出更多的努力并且还会遇到更多的困难。Java 多国语言实现设计的一个目标是扭转这种局面。Java 试图寻找比编写针对地区的代码更容易的办法编写多国语言实现代码，多国语言实现主要影响三个包：

- java.util　包含了 Locale 类。Locale 类封装了关于场所的信息，但是不提供实际的针对场所的操作。现在，可以向受到影响的方法传递 Locale 对象作为参数，该参数将改变方法的行为。如果不指定 Locale 对象，Java 将使用环境中的缺省 Locale。这个包还提供了对 ResourceBundles 和封装可移植的本地数据对象的支持。
- java.io　java.io 包里的所有类都使用 InputStreams 和 OutputStreams，都有使用 Reader 类和 Writer 类的相应类。Reader 和 Writer 类似于 Stream，除了 Reader 被设计用来处理 16 位统一码字符，而不是 8 位的字节。
- java.text　这完全是一个新包，它提供了对各种文本操作的支持。包括文本校对、格式

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

化日期和数字,以及与分析与语言有关的数据。

29.3.1 Locale 类

Locale 对象封装了与特定场所有关的信息。它由足以唯一地标识场所的地区信息组成。当场所敏感的方法接收到 Locale 参数时,它试图为特殊的场所修改它的行为。Locale 对象是使用语言代码、国家代码和一个可选的变体码来初始化的。这三个东西定义了一个地区,但是用户不需要全部指定这三个因素。例如,用户可以创建一个加利福尼亚州的美国英语对象。如果用户询问 Calender 类每年的第一个月是几月,那么 Calender 将试图查找适合加利福尼亚州的美国英语的月名。因为月名不受用户输入的州名影响,所以 Calender 类中没有内置对加利福尼亚英语的支持,因此它将试图找到一个最合适的名字。它接下来查找适合美国英语的月名,因为月名在讲英语的国家里使用的是相同的名字,所以这次查找也将失败。最后,Calendar 类返回英语 Locale 的月名。这种最适合查找过程允许程序员完全控制多国语言实现代码的各个方面。

用户可以使用下述句法创建 Locale 对象:

```
Locale theLocale = new Locale("en", "US");
```

在这里"en"代表英语,"US"代表美国。这两个双字母代码被 Java 在内部用于标识语言和国家。它们分别由 ISO—639 和 ISO—3166 标准文档定义。关于这两个文档更详细的资料用户可在下列网址上获得:

http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt

http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

目前,JDK 在它的所有场所敏感的类中,比如 Calender,和 NumberFormat 等,支持下述语言和国家组合,如表 29.1 所示。这个表在将来可能发生变化,所以用户应该注意查看最新文档。

表 29.1 由 JDK 支持的场所

场　所	国　家	语　言
da_DK	丹麦	丹麦语
DE_AT	奥地利	德语
de_CH	瑞士	德语
de_DE	德国	德语
el_GR	希腊	希腊语
en_CA	加拿大	英语
en_GB	联合王国(即英国)	英语
en_IE	爱尔兰	英语
en_US	美国	英语
es_ES	西班牙	西班牙语
fi_FI	芬兰	芬兰语
fr_BE	比利时	法语

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

场　　所	国　　家	语　　言
fr_CA	加拿大	法语
fr_CH	瑞士	法语
fr_FR	法国	法语
it_CH	瑞士	意大利语
it_IT	意大利	意大利语
ja_JP	日本	日语
ko_KR	韩国	高丽语
nl_BE	比利时	荷兰语
nl_NL	荷兰	荷兰语
no_NO	挪威	挪威语(Nynorsk)
no_NO_B	挪威	挪威语(Bokmål)
pt_PT	葡萄牙	葡萄牙语
sv_SE	瑞典	瑞典语
tr_TR	Turkey	土耳其语
zh_CN	中国	汉语(简体)
zh_TW	中国	汉语(繁体)

用户也可以创建自定义的场所，可以通过指定唯一的国家、语言和变体序列来自定义场所。多个变体可以用下划线分隔开。为了创建运行在 Windows 机器上的加利福尼亚州的美国英语变体，用户可以使用下述代码：

```
Locale theLocale = new Locale("en", "US", "CA_WIN");
```

记住：不理解这个特殊变体的方法将会试图查找最适合的匹配，在本例中可能使用“en_US”。

这里列出来的这个双字母简称并不意味着将要显示给用户；它们只是内部表示。为了显示，需要使用表 29.2 中的 Locale 方法之一。用户将会注意到这些方法一般是可覆盖的，所以用户即可以获得当前场所的参数，也可以获得指定场所的参数。

表 29.2　场所显示方法

方　法　名	描　　述
getDisplayCountry( ) getDisplayCountry(Locale)	缺省场所或指定场所的国家名
getDisplayLanguage( ) getDisplayLanguage(Locale)	缺省场所或指定场所的语言名
getDisplayName( ) getDisplayName(Locale)	缺省场所或指定场所的完整场所名
getDisplayVariant( ) getDisplayVariant(Locale)	场所的变体名。如果指定场所名未找到，该方法返回变体代码

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

当用户希望与 Locale 对象交互时这些方法是非常有用的。下面是一个使用 getDisplay-Language 方法的例子：

```
Locale.setDefault(new Locale("en", "US");//将缺省场所设置为美国英语
Locale japanLocale = new Locale("ja:, "JP");//为日本创建场所
System.out.println(japanLocale.getDisplayLanguage( ));
System.out.println(japanLocale.getDisplayLanguage(Locale.FRENCH));
```

这个代码片段打印出由 japanLocale 使用的语言名。在第一行中，它设置了缺省 Locale，缺省 Locale 通常被设成美国英语。第一个打印语句打印的内容是 Japanese。第二个打印语句显示的语言名是用法语写的，结果是 Japonais。Locale 类的所有显示方法都使用相同的方式。几乎所有的多国语言实现 API 方法都允许用户显式地控制用于场所化的方法，但是在大多数情况下用户只要使用缺省的场所就足够了。

在前面的例子中另一个需要注意的东西是静态常量 Locale.FRANCH 的使用。Locale 类提供了许多这一类的常量，其中每一个都是一个代表相应 Locale 对象的捷径。这些对象如表 29.3 所示。

表 29.3 Locale 类的静态对象

常 量 名	场 所	捷径代表的内容
CANADA	加拿大英语区	new Locale("en", "CA", "")
CANAD_FRENCH	加拿大法语区	new Locale("fr", "CA", "")
CHINA SCHINESE PRC	汉语(简体)区	new Locale("zh", "CN", "")
CHINISE	汉语区	new Locale("zh", "", "")
ENGLISH	英语区	new Locale("en", "", "")
FRANCE	法国	new Locale("fr", "FR", "")
FRANCH	法语区	new Locale("fr", "", "")
GERMAN	德语区	new Locale("de", "", "")
GERMANY	德国	new Locale("de", "DE", "")
ITALIAN	意大利语区	new Locale("il", "", "")
ITALY	意大利	new Locale("it", "IT", "")
JAPAN	日本	new Locale("jp", "JP", "")
JAPANESE	日语区	new Locale("jp", "", "")
KOREA	韩国	new Locale("ko", "KR", "")
KOREAN	高丽语区	new Locale("ko", "", "")
TAIWAN TCHINES	台湾(繁体)	new Locale("zh", "TN", "")
UK	大不列颠	new Locale("en", "GB", "")
US	美国	new Locale("en", "US", "")

29.3.2 将场所敏感数据打包

Locale 类允许用户很容易地处理场所敏感的方法。但是大部分程序(尤其是小应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

和基于图形用户界面的应用程序)要求使用具体场所的 String 对象、数据和其它资源。例如，大部分图形用户界面程序都有 OK 按钮(确认按钮)和 Cancel 按钮(取消按钮)。这些按钮对于美国场所没有任何问题，但是对于其它场所则可能要求为按钮设置不同的标签。例如，在德语中用户可能需要使用 Gut 和 Vernichten 来替换 OK 和 Cancel。传统上这些信息都被包含在应用程序的代码中，就像前面提到的 Joe Programmer 那样，当需要支持许多场所化版本时将会遇到许多问题。为了解决这个问题，Java 提供了将这些需要的数据封装到能被虚拟机在需要的时候加载的对象中。这个对象叫做 ResourceBundles。

ResourceBundles—命名规范：ResourceBundles 是一个抽象的类，必须经过扩充方能具有一定的功能。ResourceBundles 是通过类加载器的名字加载的，必须遵循严格的命名规则才能保证加载正确。举例来说，如果用户通过扩展 ResourceBundles 创建了一个名为 LabelBundle 的类，这个名叫 LableBundle 的类提供了缺省信息；LabelBundle _ fr 提供了法语标签；LabelBundle _ ge _ GE 提供了德语标签；LabelBundle _ en _ US _ MAC 提供了 Macintosh 系统上使用的美国英语标签。用户可以使用下述静态方法申请 ResourceBundles：

```
ResourceBundle getResource Bundle(String baseName, Locale locale ClassLoader loader)
```

该方法使用了指定的 ClassLoader 去查找与 baseName 匹配的类和指定的 Locale 的属性。一种非常特殊的查找方式用于查找与用户申请的 Bundle 最相近的匹配：

```
bundleName +"-" + localeLanguage +"-"+ localeCountry +"-" + localeVariant
bundleName +"-" + localeLanguage +"-"+ localeCountry
bundleName +"-" + localeLanguage
bundleName +"-" + defaultLanguage +"-" + defaultCountry -"-" + defaultVariant
bundleName +"-" + defaultLanguage +"-" + defaultCountry
bundleName +"-" + defaultLanguage
bundleName
```

在我们的例子里如果用户使用 fr _ FR _ WIN(法语，法国，Windows 平台)并用 LabelBudle 作为基本名，那么 getResourceBundles( )方法将执行下列步骤：

1. 查找 LableBundle _ fr _ FR _ WIN 类，这个查找肯定会失败，因为用户没有定义这个类。
2. 查找 LableBundle _ fr _ FR，也肯定会失败，因为用户也没有定义法国法语包。
3. 查找 LableBundle _ fr。这一查找将获得成功并返回使用空上名字的类。但是如果这个查找也失败(如果用户没有提供法语包)，那么查找工作将继续进行，将使用由缺省 Locale 提供的语言，国家和变体代码。

创建 ResourceBundle　既然用户已经理解了使用 ResourceBundle 的命名规范，那么现在让我们来看一下如何创建它们。ResourceBundles 最简单的形式是直接扩展 ResourceBundle 类并覆盖一个方法：

```
Object handleGetObject(String key)
```

这个方法返回一个响应指定线索的对象。这些线索是在内部对存放在 ResourceBundle 中的内容的描述，并且这些线索对于某些数据的所有具体场所化版本是相同的。用户的 LableBundle 的极其简化的版本可能如下所示：

```
class Lable extends ResourceBundle{
    public Object handleGetObject(String key){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        if (key.equals("OK"))
            return "OK";
        else if (key.equals("Cancel"))
            return "Cancel"
        //其它标签可能在这里处理
        return null;//如果线索不匹配,总是返回 null
    }
}
```

同一个包的其它版本可能返回翻译成其它不同语言的值。但是用户可以看到,当用户有较多的线索时这种处理线索值对的方法不是非常有效的。幸运的是,Java 提供了两个 ResourceBundle 的子类可以使这种处理变得更加容易;这两个子类是:ListResourceBundle 和 PropertyResourceBundle。

ListResourceBundle 使用一个二维数组存放前面使用过的索引对,这个二维数组每一行只有两个元素即该二维数组只有两列。所有需要用户做是事情是覆盖缺省的 getContents( )方法,像这样:

```
class LableBundle extends ListResourceBundle{
    static final Object[ ][ ] labels = {
                                    {"OK", "OK"},
                                    {"Cancel", "Cancel"},
                                    {"Anotherkey", "Another Value"}
                                    //在这里可以写上更多的线索值对
    };
    public Object[ ][ ] getContents( ){
        returns labels;
    }
}
```

用户也可以使用散列表来提供自己的类似功能,但是这很费事,除非用户希望动态地修改显示内容才值得这样做。

PropertyResourceBundle 是根据存放在磁盘上的预定义"属性"文件的需要创建的。这些对象通常用于系统设置,或者在大量的数据需要存放在索引值对中时才使用。PropertyResourceBundle 从同名的相应类文件中生成,但使用的扩展名为 Properties。为了实现 LableBundle_de_DE 类,用户可以使用下列内容提供一个名为 LableBundle_de_DE.properties 的文件:

```
OK = Gut
Cance = Vernichten
Anotherkey = 这个值包许多文本。当然这些文本首先是被翻译成德文的…
```

内容总是以"key = 值"形式指定的,并且是被假定为 String 对象的(虽然它们可以被强制为其它适当的对象)。这个功能是基于 java.util.Properties 类的。用户可以参考第 47 章"java.lang",在第 47 章中用户可以更多地了解 java.util 包。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

注意：虽然此处给出的例子全部处理的是 String 对象，但是 ResourceBundle 可以存储任何类型的对象，包括 Date(日期型)、Applet、GUI 元素，甚至是其它的 ResourceBundle。

访问 ResourceBundle　正像前面提到的那样，用户可以通过静态的方法 getResourceBundle( )使用名字来加载 ResourceBundle。假设加载成功(否则它将抛出异常)，那么用户就可以使用 getObject( )方法查询该包中的单项值。当然，需要将其显式地强制为用户希望的对象种类，为此用户需要事先知道这方面的信息。为了方便起见，ResourceBundle 还提供了可以返回已经强制了类型的对象的下述方法：

- getMenu(String)
- getMenuBar(String)
- getObject(String)
- getString(String)
- getStringArray(String)

29.3.3　java.util 包中的其它与多国语言实现有关的部分

多国语言实现要求对 java.util 类作另外两个修改。例如，在 Java1.0 中 Date 类用于时间操作，那时它只是一个对某个特定瞬间的封装。现在为了创建 Date 对象，用户应该使用 Calender 类。Calender 是一个抽象类，它提供与文化无关的处理时代、世纪、年、月、日和时间的各种方法。为了实例化 Calender 方法，用户必须通过扩展 Calender 来提供基于特定的 Calender 标准的方法。唯一一个来自于 JDK 的(迄今为止)类是 GregorianCanlendar，该类为世界上最流行的日历系统提供了非常复杂的功能。将来的版本可能还会包括对阴历、季节或其它日历系统的支持，与 Calender 类的相对应的类叫做 TimeZone(和 SimpleZone)，它通常不被程序员使用，它允许日期和时间针对其它的时区做一些调整。

Date、Calendar 和 TimeZone 类提供了数量巨大的功能，以至于许多程序员根本不需要了解其中的许多功能。用户不需要理解使用这些类的时间算法的复杂细节；这些类都包含允许用户获得当前时间和日期与以场所敏感的方式显示时间和日期的方法。通过使用由这些类提供的方法，就可以使得用户的程序缺省地具体场所化了，用户本人不必做什么额外的工作。

注意：这几个类还有许多方法，如果用户感兴趣，用户可以看一下本章末尾的 Calender 类与 Date 类交互的例子。要了解更详细的内容，用户可以直接查看 Java 应用编程接口(API)文档。

29.4　多国语言实现的输入和输出(I/O)

本来，java.io 包全部是操作在字节流上的，即 8 位不间断序列上。但是 Java 的统一码字符是 16 位的，这使得它难以使用字体流。因此，如果用户查看 java.io 包可以发现它全部使用了 16 位字符流 Reader 和 Writer，它们是与老的 InputStream 和 OutputStream 相对应的。这两套类即可以共同工作也可以独立工作，具体用法取决于用户是否需要处理多种文本的输入和

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

输出。

29.4.1 字符集转换器

字符以二进制表示的方法叫做编码方案。英语文本使用的最常见的方案叫做 ISO Latin-1 编码。被某种编码支持的字符的集合被称为该编码的字符集，它包括所有由该编码方案描述的可能字符。通常，编码方案的前 127 个字符与通常的 ASCII 字符集相应，这 127 个字符包括所有的标准字符和发音标志。但是，许多编码会完全不是如此对应，特别是像汉语编码和日语编码使用的字符集与英语几乎没有相似的地方。

幸运的是 Java 1.1 为处理最常见的编码提供了一些类。ByteToCharConverter 类和 CharToByteCouverter 类负责对程序中用到的文本与由 Java 支持的标准的统一码字符之间的转换。每种编码方案都以标签的形式给出，标签是一种标识。Java 1.1 支持的编码方案和它们的标签全部列在 29.4 中。

表 29.4 JDK 1.1 支持的字符编码

标 签	编码方案描述
8859-1	ISO Latin-1 方案
8859-2	ISO Latin-2 方案
8859-3	ISO Latin-3 方案
8859-4	ISO Latin-4 方案
8859-5	ISO Latin/Cyrillic 方案
8859-6	ISO Latin/Arabic 方案
8859-7	ISO Latin/Greek 方案
8859-8	ISO Latin/Hebrew 方案
8859-9	ISO Latin-5 方案
BigS	大五码中文繁体方案
CNS11643	CNS 11643 中文繁体方案
Cp1250	Windows Eastern Europe/Latin-2 方案
Cp1251	Windows Cyrillic 方案
Cp1252	Windows Western Europe/Laatin-1 方案
Cp1253	Windows Greek 方案
Cp1254	Windows Turkish 方案
C p1255	Windows Hebrew 方案
Cp1256	Windows Arabic 方案
Cp1257	Windows Baltic 方案
Cp1258	Windows Vietnamese 方案
Cp437	PC Original 方案
Cp737	PC Greek 方案
Cp775	PC Baltic 方案
Cp850	PC Latin-1 方案
Cp852	PC Latin-2 方案
Cp855	PC Cyrillic 方案
Cp857	PC Turkish 方案
Cp860	PC Portuguese 方案
Cp861	PC Icelandic 方案
Cp862	PC Hebrew 方案
Cp863	PC Canadian French 方案
Cp864	PC Arabic 方案

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

标　签	编码方案描述
Cp865	PC Nordic 方案
Cp866	PC Russian 方案
CP869	PC Modern Greek 方案
Cp874	Windows Thai 方案
EUCJIS	Japanese EUC 方案
GB2312	GB2312-80 简体中文方案
JIS	JIS 方案
KSC5601	KSC5601 Korean 方案
MasArabic	Macintosh Arabic 方案
MacCentraEurope	Macintosh Latin-2 方案
MacCroatian	Macintosh Croatian 方案
MacCyrillic	Macintosh Cyrillic 方案
MacDingbat	Macintosh Dingbat 方案
MacGreek	Macintosh Greek 方案
MacHebrew	Macintosh Hebrew 方案
MacIceland	Macintosh Iceland 方案
MacRoman	Macintosh Roman 方案
MacRomania	Macintosh Romania 方案
MacSymbol	Macintosh Symbol 方案
MacThai	Macintosh Thai 方案
MacTurkish	Macintosh Turkish 方案
MacUkraine	Macintosh Ukraine 方案
SJIS	PC and Windows Japanese
UTF8	Standard UTF-8 方案

Java 还为开发人员提供了创建自己的编码的手段，也提供了为已存在的却不被支持的编码创建转换器的手段。字符转换的细节相当复杂，感兴趣的用户可以直接查看 Java 的网页。

29.4.2　Reader 和 Writer

字符流大量地使用了字符集转换器。幸运的是，它们隐藏了底层复杂的转换细节，这样使得开发人员即使不知道多国语言实现的过程也可以很容易地编写 Java 程序，并且这些程序被缺省地实现了多国语言支持。

使用字符流比起使用字节流有更多的优点，虽然它们在字节读取的顶层事先已经增加了字符转换，但是它们仍然允许使用非常有效的缓冲区技术。字节流被设计成每次只读取一个字节的信息，而字符流则可以一次读入一个缓冲区。Sun 公司宣称，和有效的锁定方案结合使用可以弥补由转换处理引起的速度损失。在老的类层次上的输入和输出流现在都有一个相应的 Reader 类和 Wtrter 类可以执行类似于字符流的功能(如表 29.5 所示)。

表 29.5　输入/输出流和相应的 Reader 类与 Writer 类(Sun MicroSystem 公司的)

字节流类 (InputStream/OutputStream)	相应的字符流类 (Reader/Writer)	功　　能
InputStream	Reader	所有其它类都从中继承方法的抽象类
BufferedInputStream	BufferedReader	为输入操作提供缓冲区
LineNumberInputStream	LineNumberReader	跟踪行号
ByteArrayInputStream	CharArrayReader	从数组中读取数据

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

字节流类 (InputStream/OutputStream)	相应的字符流类 (Reader/Writer)	功　能
N/A	InputStreamReader	将字节流翻译成字符流
FileInputStream	FileReader	允许从磁盘文件上输入
PushbackInputStream	PushbackReader	允许字符被压回流中
PipedInputStream	PipedReader	从管道中读取数据
StringbufferInputStream	StringReader	从字符串中读取数据
OutputStream	Writer	字符输出流的抽象类
BufferedOutputStream	BufferedWriter	缓冲区输出,使用平台的行分隔符.
ByteArrayOutputStream	CharArrayWriter	向字符数组中写数据
FilterOutputStream	FiterWriter	过滤字符输出的抽象类
N/A	OutputStreamWriter	将字符流翻译成字节流
FileOutputStream	FileWriter	将字符流翻译成字节文件
PrintStream	PrintWriter	向 Writer 打印值和对象
PrpedOutputStream	PipedWriter	向 PipedReader 写入数据
N/A	StringWriter	向字符串中写入数据

如果用户正在开发新程序,那么这种改变带来的冲击是非常小的。所有需要用户做的事情就是记住在以前使用 InputStream 和 OutputStream 的位置上使用 Reader 和 Writer。用户需要关心的最大改变是与 DataInputStream 和 PrintStream 有关的改变,它们是两个选择发送文本输入和输出的两个类。DataInputStream.readLine( )方法已不再提倡使用了,用户应该使用 BufferedReader.readLine( )方法。进一步地说,用户不再需要实例化新的 PrintStream 对象了,尽管用户仍然为了调试程序而使用事先已经存在了的 PrintStream(比如 System.out)的方法。为了输出行结束字符串,用户应该使用 PrintWriter 类。主要的变化是用于和 DataInputStream 类与 PrintStream 类通信的代码(包括套接字、文件和管道的代码)必须使用适当的 Reader 类和 Writer 类修改。为了使修改更加容易,Java 提供了一个名为 InputSteamReader 的类和一个 OutputSteamWriter 的类,它们用于基于字节流创建新的 Writer 或 Reader。这使得 Reader/Writer 系统与所有当前使用字节流的其它类可以兼容(比如 URL、套接字和文件等)。

29.5 新的 java.text 包

最先进和最复杂的多国语言实现应用编程接口(API)的特性由 java.text 包提供。它们包括了许多与语言无关的文本格式和文本组织。例如,日期格式对程序员来说是经常发生问题的,在美国,日期是按照月 - 日 - 年顺序给出的,但是在欧洲,日期则是按照日 - 月 - 年的顺序给出的。这使得解释 10/2/97 变得困难:它代表的是 1997 年 10 月 2 日,还是 1997 年 2 月 10 日呢? 这就是格式化文本属性的用途。这些类的大部分不需要直接实例化并且可以通过静态的 getDefault( )方法访问。

另一方面,文本对照是一种根据特定规则分类文本的处理。在英语中,按字母顺序分类相

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

当容易，因为英语没有特殊的字符(比如重音)。如果有重音符号，那么分类就困难了。在法语中，分类就不那么简单，有的两个词非常相似(像 péché 和 pêche)但却有完全不同的含义。这时哪一个单词应该排在前面呢？另外像标点符号和连字符这一类字符又该怎么办？Java 的 Collation 类提供了较好的针对语言排序标准的手段。

在语言中文本的界线也是模糊的。单词、语句和段落从何处开始到何处结束？在英语中，圆点一般标志着语句的结束，但是它总是代表这个含意吗？当然不是。TextBoundary 类和 CharacterIterator 类可以基于语言自身的标准将文本智能地分解成各种子单元。Java 内置了对一些语言的支持，但是用户总是可以定义自己的规则集，TextBoundary 返回字符串中出现的界限的整数索引值。下述代码片段将字符串分解成单词：

```
String str = "This is a line of text. It contain many words ,setence , and formmatting.";
TexBoundary byWord = TextBoundary.getWordBreak( );
int from , to ;
from = byWord.first( );
while((to = byWord.next()) ! = DONE){
System.out.println(byWord.getText( ).substring(from,to));
   from = to
}
```

这个代码片段按照每个单词一行的格式打印出 str 字符串。虽然这个例子本身并不重要，但是文本界限却是非常重要的，特别是在要求文本选择和智能的单词封装等图形用户界面应用程序中。

29.6　示例：InternationalTest

为了更好地理解这些多国语言实现的特性如何协同工作，让我们看一下非常简单的 Java 应用程序示例。该应用程序使用了本章中讨论的几个特性。该程序包含在本书的附带光盘中，如果用户喜欢自己运行它，可以从光盘中找出该应用程序。

这个应用程序非常简单。它使用了三个指定场所的命令行参数。它使用这三则信息做下述事情：

1. 显示关于缺省场所和用户输入的场所信息。
2. 试图加载与指定场所相应的 ResourceBundle 并打印出该 Bundle 包含的信息。
3. 显示日期，具体场所化为指定的场所。

除了 InternationaTest 类以外，该程序要求几个其它的类。最重要的是 ResourceBundle 类，它响应不同的场所(目前，ResourceBundle 必须作为公共类来创建，但是在将来的 JDK 版本中这一点可能会改变)。另一个需要注意的东西是这个小应用程序传递了一个"空"的 Classloader 参数给 getResourceBundle( )方法，并且没有显式的 ClassLoader。只要 ResourceBundle 可以使用 CLASSPATH 变量，那么用户就不需要单独的 ClassLoader 去加载它们。另一方面如果用户正在编写小应用程序，那么用户就需要一个 ClassLoader 对象从国际互联网上加载类。用户也可以使用相同的 ClassLoader 实例加载小应用程序，像这样：

```
ClassLoader loader = this.getClass( ).getClassLoader( );
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 29.1 是 InternationalTest 程序的完整列表。

程序清单 29.1 InternationalTest.java

```
import java.util.*;
import java.lang.*;
import java.text.DateFormat;

class InternationalTest extends Object {

public static void main(String args[]) {

String lang = "", country = "", var = "";

   try {
      lang = args[0];
      country = args[1];
      var = args[2];
   } catch(ArrayIndexOutOfBoundsException e) {
      if( lang.equals("") ) {
         System.out.println("You must specify at least one parameter");
         System.exit(1);
         }
   }

   Locale locale = new Locale(lang, country, var);
   Locale def = Locale.getDefault();

   System.out.println("Default Locale is: " + def.getDisplayName() );
   System.out.println("You have selected Locale: " + locale.getDisplayName() );
   System.out.println("Default language, localized for your locale is: " +
            def.getDisplayLanguage( locale ) );
   System.out.println("Default country name, localized: " +"
                        locale ) );

   ClassLoader loader = null;

   ResourceBundle bundle = null;
   try {
      bundle = ResourceBundle.getResourceBundle( "TestBundle", locale, loader );
   } catch( MissingResourceException e) {
      System.out.println( "No resources available for that locale." );
   } finally {
         System.out.println( "Resources available are:");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            System.out.println(" r1:" + bundle.getString("1") );
            System.out.println(" r2:" + bundle.getString("2") );
        }

        DateFormat myFormat = DateFormat.getDateTimeFormat ( DateFormat.FULL, DateFormat.FULL,
locale);
        Calendar myCalendar = Calendar.getDefault( locale );
        System.out.println("The localized date and time is: " +
                myFormat.format( myCalender.getTime() ) );

        }
    }
```

图 29.1、29.2 和 29.3 显示了 InternationalTest 程序的输出。

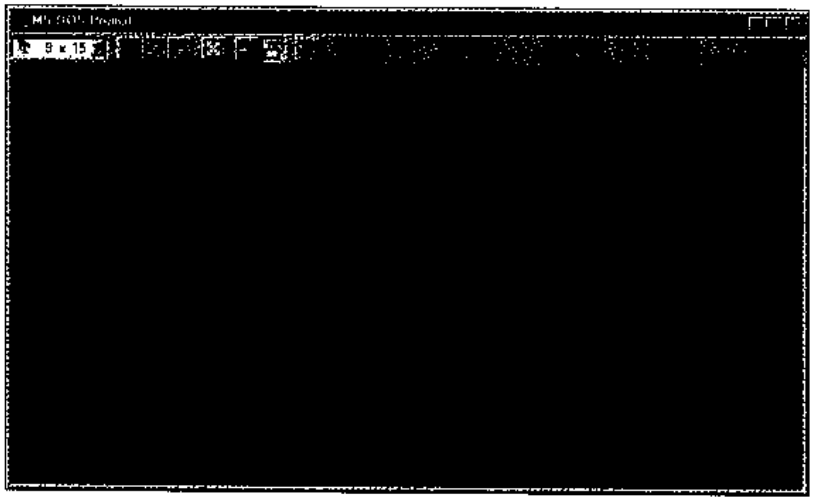


图 29.1　美国英语场所

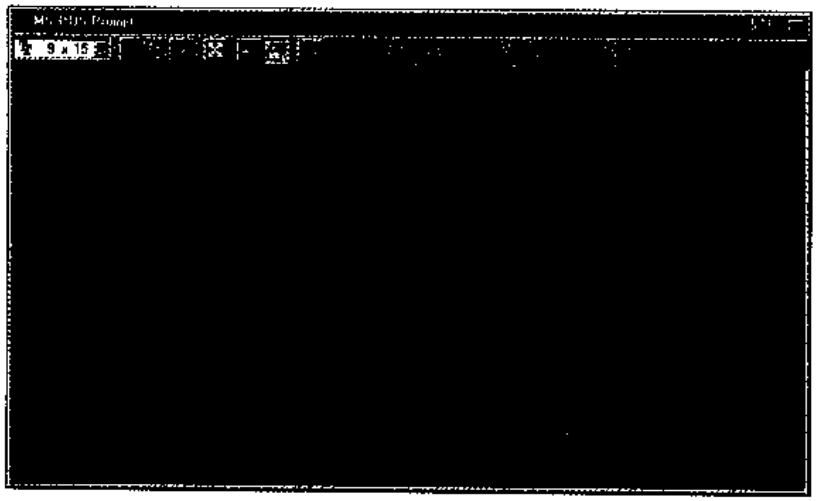


图 29.2　加拿大法语场所和加拿大法语的 Machintosh 场所

那么留给 Joe Programmer 做的事情在哪里？他需要做的事情是将他的应用程序转换成支持多国语言实现应用程序编程接口的应用程序。他的表、文本和场所化资源需要被封装在

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

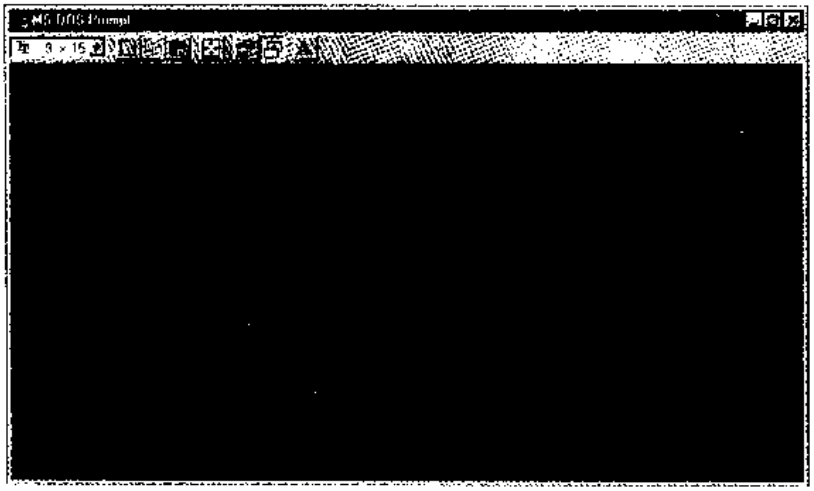


图 29.3 加拿大英语场所和德国德语场所

他想支持的每个场所的 ResourceBundle 中。他还需要调整几个使用具体场所的日期、时间和信件格式的方法和对象。当 Joe Programmer 做完这些工作之后他会发现，他的程序不仅具体化到了各个场所，而且不再需要支持同一程序的多个版本了。甚至，当有新的场所需要支持时，他不必修改任何源代码，只需把面向场所的资源翻译成新的语言或习惯就可以。现在 Joe Programmer 的程序真正成为可移植、可重用和平台无关的程序了。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 30 章　通信和网络

本章内容：

30.1　TCP/IP 概述
30.2　TCP/IP 协议
30.3　统一资源定位器(URL)
30.4　Java 和 URL

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

30.1 TCP/IP 概述

尽管还有许多优点，但是 Java 与国际互联网网络的强有力集成使得 Java 极为迅速地被计算界接受。国际互联网革命从根本上改变的个人计算机的使用方式，使得单独的计算机结合在一起，发布和共享数以百万计的个人计算机上的巨大资源。建立在这个基础上的 Java 将为计算界带来另一场重要的革命。

Java 运行环境被设计成允许应用程序很容易被编写得能够与远程系统通信和共享处理能力。这些功能中的许多是由 Java 的 java.net 包里的标准 Java 应用编程接口提供的。

本书中共有四章讲解和使用 java.net 包中的类，本章是其中的第一章。作为这方面讨论的基础，本章将重点讲解国际互联网网络协议集—TCP/IP。

TCP/IP 是一套国际互联网上各种系统之间互联时使用的协议。TCP/IP 为各种不同的硬件提供了通用的编程接口。这套协议支持使用不同网络介质的物理独立的网络连接。TCP/IP 使得世界各地各式各样的孤立网络联系成国际互联网成为可能。

模型是对工作系统非常有用的抽象描述，它忽略了细节使得人们能够看清楚宏观上的交互作用。模型可以让我们更好地理解和扩展一个系统。理解网络通信是学习 TCP/IP 基础理论的必要准备。

30.1.1 OSI 参考模型

网络系统通常使用名为开放系统互联(Open Systems Ineterconnect 简称 OSI)参考模型的网络协议结构来描述。OSI 方案是国际标准化(ISO)组织的一项更大的工程中的一部分。OSI 协议从未像 TCP/IP 协议那样成功，但是 ISO 的模型却为互联网作出了不朽的贡献。

该模型是由七个提供不同功能的层组成，每个层都定义了一些特性，合在一起就可以实现网络通信。每个分层模型的软件实现被称作协议堆栈。

OSI 模型如图 30.1 所示。用户应用程序插入某个层或每个层，封装数据直到最后一层。当信息传递到目标机时，信息再由下至上地被层层还原为原来的信息。

下面的七个层都扮演着各自的角色，每个层的功能都相互独立，但彼此之间又密切相关。

- 应用层　包含人们实现交互的网络应用，比如电子邮件、文件传输和远程登录。
- 表示层　创建普通的数据结构
- 会话层　管理网络应用程序之间的连接
- 传输层　保障接收数据与发送数据一致
- 网络层　按规定方式经过各种物理网络向已知宿主机上发送数据。
- 数据链路层　通过统一的物理网络可靠地传输和接收信息包。
- 物理层　定义网络的物理属性，比如电压，电缆类型、接口的针数。

30.1.2 TCP/IP 网络模型

OSI 模型对于理解 TCP/IP 通信结构很有帮助。当作为分层的模型来看时 TCP/IP 通常可以被视为由四个层次组成：

- 应用层
- 网络层

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

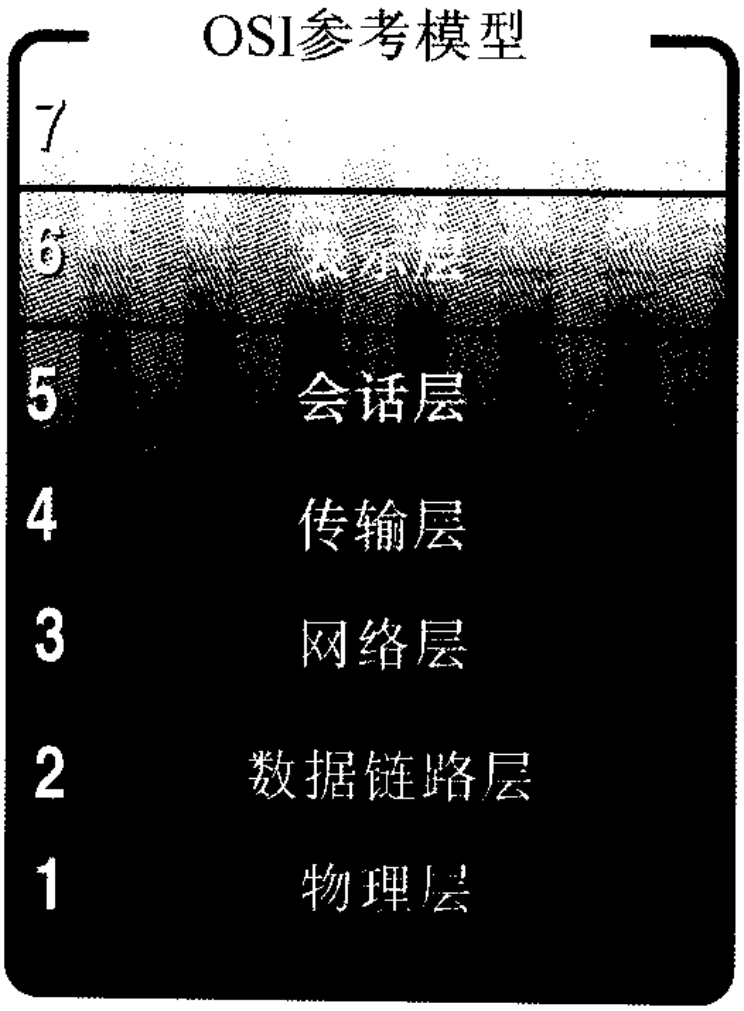


图 30.1　OSI 参考模型由七个层次组成

- 传输层
- 链路层

这四层的层次结构如图 30.2 所示。这种层次关系和 OSI 的模型之间很难找到一个确定的映射关系，所以在本章中我们也不作这种努力。

像 OSI 模型一样，每一个 TCP/IP 层都有不同任务，在接下来的四个部分里我们具体地描述这四个层。

应用层　网络应用程序依靠一个清晰的对话定义。在客户服务器系统中，客户应用程序知道如何请求服务，并且服务器也知道如何适当地响应。实现该层的协议包括 HTTP、FTP 和 Telnet 等。

传输层　传输层允许网络应用程序通过定义明确的通道及某些特性获取信息。TCP/IP 中实现该层的两个协议是传输控制协议(TCP)和用户数据协议(UDP)。

网络层　网络层允许将数据向相邻的 TCP/IP 网络上的任一宿主机上传递，而不管每个网络在物理上的差别。网间协议(IP)是在该层中传输数据的机制。

链路层　链路层由用于向同一个物理网络上传输数据的低级协议组成。非 TCP/IP 协议比如 Ethernet、TokenRing、FDDI 和 ATM 实现了这一层。包括在这些层中的数据是使用普通机制封装的：协议拥有一个头，用于标识下述元信息，比如源、目标和其它属性，另外协议还有

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

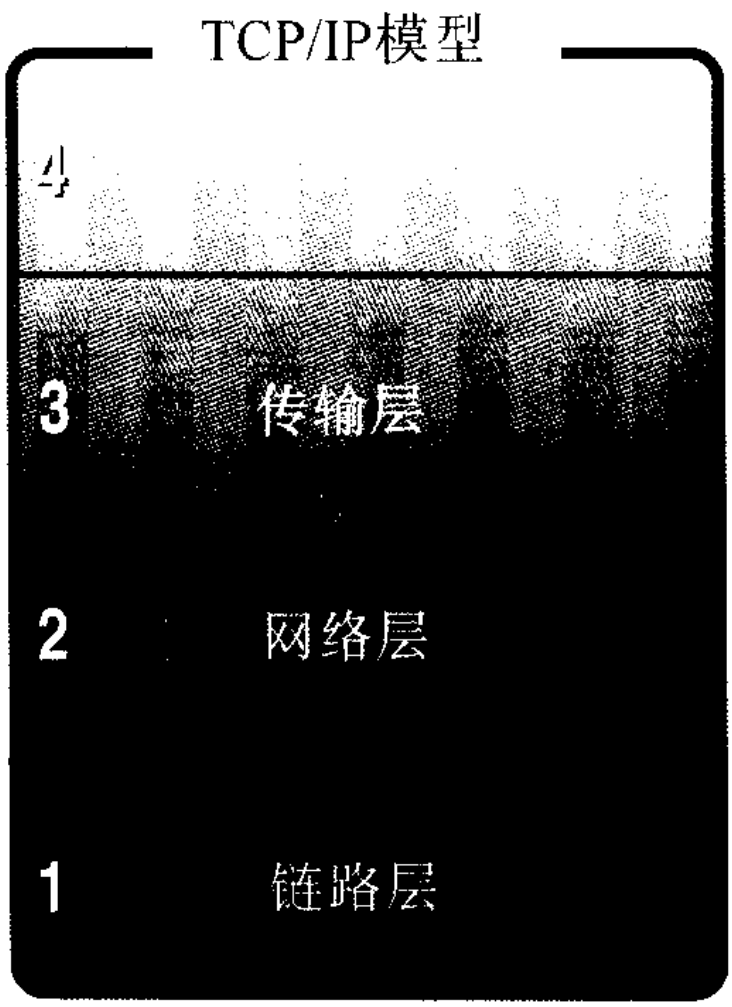


图 30.2 TCP/IP 模型可以分为四个层

包含真实信息的数据部分。从顶层的协议开始封装在底层的数据直至所有协议被封装完毕，在传递到目的后，从底层协议向上，一层层地将协议堆栈中各层协议的信息分开恢复以得到原来的数据。图 30.3 解释了这种封装的概念。

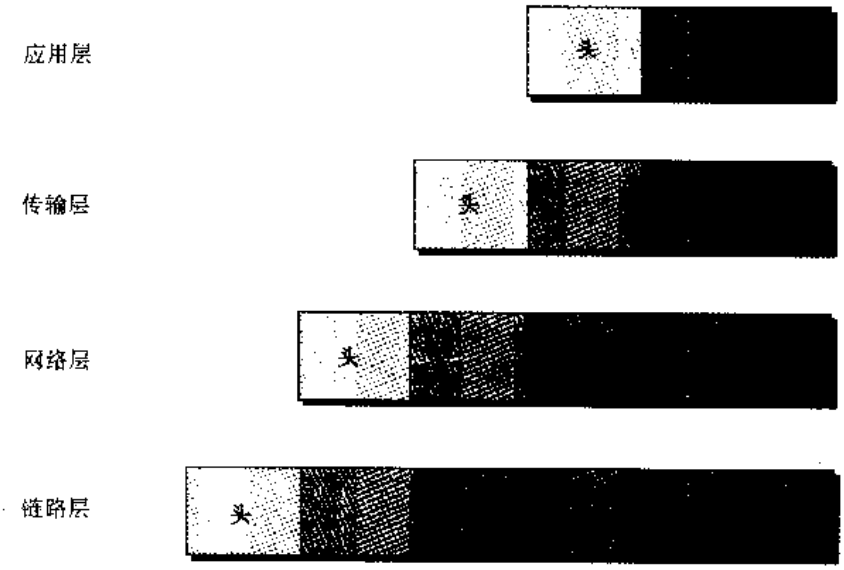


图 30.3 当数据通过 TCP/IP 的各个层时是被封装起来的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

30.2　TCP/IP 协议

TCP/IP 方案中有三个最常用的协议，我们可以详细了解它们的属性。理解这三个协议（IP、TCP 和 UDP）之间的交互对开发网络应用程序是至关重要的。

30.2.1　Internet 协议（IP）

IP 是 TCP/IP 的基础。互联网上的所有数据都流经 IP 包，IP 包是传输中的最基本单元。IP 是无连接、不可靠的协议。作为无连接协议，IP 在将数据传送到远地系统之前，IP 并不交换控制信息，IP 包只是被发送到目的地，而且这些 IP 包被认为将会被正确处理。说 IP 是不可靠协议是因为它对丢失的 IP 包并不重发，也不检查数据的完整性。这些任务是由高层协议完成的，如 TCP。

IP 定义了一个通用的编址方案，叫做 IP 地址。IP 地址是一个 32 位数，在国际互联网上的每个标准地址是唯一的。任意一个 IP 包根据其包头中定义的 IP 地址信息，可以被传输到目标机器上。IP 地址写作四个 0 到 255 之间的数，每个数之间使用圆点分隔（例如 124.148.157.6）。

一个 32 位的二进制数对计算机而言是一个很好的地址形式。但是对于人而言则是很难理解和记忆的。于是有人开发了域名系统（DNS），将 IP 地址映射为更容易记忆的标识符，也将这些标识符映射为 IP 地址。用户可以使用 www.netspace.org 代替 128.148.157.60。

需要强调的是 IP 并不使用域名也不理解域名。当一个应用程序向另一台连到互联网上的机器发送数据时，它首先用 DNS 将域名翻译成 IP 地址。而接收程序反向翻译，使用 DNS 返回 IP 地址，IP 地址和域名的关系并不是一一对应的：一个域名可以映射多个 IP 地址，多个 IP 地址可以映射到同一个域名上。

警告：尤其重要的一点是 DNS 数据并不可信，世界上许多不同的系统中都维护 DNS 记录。一个 DNS 服务器可能被修改，可以使用虚伪信息进行设置等。事实上，在早期的 Java 实现中，由于轻信 DNS 曾导致了安全漏洞。

30.2.2　传输控制协议（TCP）

大部分国际互联网应用程序都使用 TCP 实现传输层。TCP 提供可靠的、面向连接的连续流协议。这些特性意味着：

- 可靠　当 TCP 传输的最小单元 TCP 段丢失或破坏时，TCP 能检测出来并重发必要的 TCP 段。
- 面向连接　在开始通信之前，TCP 通过给远程系统发送控制信息来建立连接，这种方式又称为握手，在连接结束后，通过一个类似的握手结束传递。
- 连续流　TCP 提供一个能够平滑地接受和发送大量数据的通信媒介，当连接建立起来后，TCP 段将为应用层提供连续的数据流。

因为上述特点，大多数国际互联网应用程序都使用了 TCP。用 TCP 能非常轻易地建立一个网络应用程序，用户不必担心数据损坏和纠正错误编码问题。但是 TCP 要求大量的代码，而且用户需要写一些额外的代码来保障可靠传输的效率。重发丢失的数据并非对所有应用程

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

序都是合适的,因为很可能那时那条信息已经过时了。在这种情况下,TCP 就不如下面将要介绍的"用户数据报协议"了。

TCP 定义的一个重要的编址方案是端口。端口将运行同一个系统中的各种 TCP 通信流分开。服务器应用程序等待 TCP 客户初始化连接,并且建立一个特殊的端口,通信以此端口为起点。这些概念在编程抽象中称为套接字。

可参考第 31 章的"TCP 套接字基础"部分。

30.2.3 用户数据报协议(UDP)

UDP 是一个比 TCP 开销小的机器之间的通信方法。和 TCP 正好相反,UDP 有下列特性:

- 不可靠 UDP 没有错误检查机制,也不重发丢失或损坏的数据。
- 无连接 UDP 在开始传送数据以前并不握手,数据发送时总设想另一方正在倾听。
- 面向消息 UDP 允许应用程序在 UDP 数据内发送自包含消息,应用程序必须在一个数据报中将所有的消息打包。

对于某些应用程序而言,UDP 比 TCP 更适合。例如对网络时间协议(NTP),丢失的指示时间数据在重新传输时的是无效的。在局域网(LAN)环境中,网络文件系统(NFS)可以更有效地为应用程序层提供可靠性,因此 NTF 使用的是 TCP。

和 TCP 一样,UDP 也提供了端口机制,这样许多应用程序可以同时发送和接收数据报。UDP 端口与 TCP 端口是完全不同的。例如当一个应用程序可以响应 UDP 的 512 端口时,另一个无关的服务程序可以处理 TCP 的 512 端口。

30.3 统一资源定位器(URL)

IP 地址唯一地标识了国际互联网上的各个系统,而端口标识了某个系统上的 TCP 服务和 UDP 服务。URL 在应用程序层上提供了统一的标识方案。使用网页浏览器的人一般都熟悉 URL,虽然它们完整的语法并不是含义十分明显的表达。开发 URL 的初衷是建立一个网络上通用的资源标识格式。但是它设计得非常通用,甚至能包含几十年以后的网络应用。相似地,URL 语法的灵活性足以包容未来的含义。

30.3.1 URL 语法

URL 分类主要依靠方案,通常由方案响应应用程序协议。方案包括 HTTP、FTP、Telnet 和 Gopher。其余的 URL 语法是依赖于方案的。这两部分信息是用冒号分开的:

```
Scheme-name:scheme-info
```

按照这种格式,mailto:dwb@netspace.org 表示"发一封电子邮件到地址为 netspace.org 机器上的'dwb'邮箱上"。ftp://dwb@netspace.org 表示打开一个与 netspace.org 的 FTP 连接并且使用 dwb 帐号登录。

30.3.2 一般 URL 格式

大多数 URL 遵循下面这样的格式:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Scheme-name://host:port/file-info # internal-reference

Scheme-name 是一个 URL 方案，比如 HTTP、FTP 或 Gopher。Host 是远程系统的域名或 IP 地址。port 是用于监听服务的端口号；因为大部分应用程序协议定义了标准端口，除非使用的是非标准端口，所以 port 和用于将 host 和 port 分开的冒号":"是可以省略的；file-info 就是远程系统所要求的资源，通常是一个文件。该文件实际可以执行服务器程序，它通常包含指向远程系统上的某个文件的路径。internal-reference 是一个 HTML 页中命名锚的标识，命名锚允许链向 HTML 页中某个特定的位置。通常情况下这部分是用不到的，这部分和分隔符"#"都可以省略掉。

这种格式是一种常用的省略格式。有关 URL 的完整信息可以在下述网页中查阅：

http://www.netspace.org/users/dwb/url-guide.html

30.4　Java 和 URL

Java 提供了开发网络客户端应用程序非常有力的机制，允许用户使用较少的语句去获得国际互联网上的资源。java.net 包中包含了 URL 类和 URLConnection 类，它们具有这种强大的功能。

> **注意**：Java 浏览器的安全管理机制通常禁止小应用程序和下载该小应用程序的机器以外的机器建立连接，这种安全特性极大地限制了小应用程序的功能，而 Java 应用程序则没有这种机制。

30.4.1　URL 类

URL 类允许用户很容易地创建包含访问远程资源的所有必要信息的数据结构。一旦建立了 URL 对象以后，用户就可以使用通用的格式获得 URL 的各个部分。URL 对象允许用户获取远程对象。

URL 类有下述四个构造器：

```
public URL(String spec) throws MafformedURLException;
public URL(String protocol, Sting host, String file) throws MalformedURLException;
public URL(String protocol, String host, int port, String file) throws MalformedURLException;
public URL(String context, String spec) throws Malformed URLException;
```

第一个构造器是最常用的构造器，它允许用户使用简单的声明创建 URL 对象，例如：

```
URL myURL = new URL("http://www.yahoo.com/");
```

第二个构造器和第三个构造器允许用户明确地指定 URL 的各个部分。最后一个构造器允许用户使用相对的 URL。相对 URL 只包含部分 URL 语法；其余的数据由和该资源相关的 URL 定义。这种情况在 HTML 页中经常见到，在 HTML 页中经常可以看到简单的 more.html 引用，它表示从当前文件所在的机器上相同的目录中获取更多的 html 文件。下面是这种构造器的示例：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
URL firstURLObject-new URL ( "http://www.yahoo.com /" );
URL secondURLObject = new URL("http", "www.yahoo.com", "/");
URL thirdURLObject = new URL("http", "www.yahoo.com", 80,"/");
URL fourthURLObject = new URL(firstURLObject, "text/suggest.html");
```

前三个语句都创建了引用 Yahoo! 主页的 URL 对象,第四个语句创建了一个 Yahoo 主页的相对引用"text/suggest.html"(即:http://www.yahoo.com/text/suggest.html)。这四个构造器全部都可以抛出 MalformedURLException 异常,用户可以在程序中捕捉该异常。程序清单 30.1 将详细地阐述这一点。注意,一旦用户建立了 URL 对象,用户就可以访问它指示的资源。为此用户必须创建一个新的 URL 对象。

30.4.2 连接到 URL 上

当用户创建了 URL 对象之后,用户可能希望实际地获取某些有用的数据。获取数据有两种途径:直接读取 URL 对象和从 URL 对象中获取 URLConnection 实例。

直接从 URL 对象中读取数据只需要很少的代码,但是这种做法缺少灵活性,并且它只允许只读连接。而许多网页服务允许用户写入传给服务器的信息,因此这种限制对用户来说很不方便。URL 类有一个 openStream( )方法,该方法可以返回 InputStream 对象,远程资源可以通过该对象逐字节地被用户读取。

逐字节地处理数据比较麻烦,所以用户常常需要将返回的 InputStream 嵌入在 DataInputStream 对象中,该对象允许用户逐行地读取信息。在修饰器程序中通常使用这种编码策略,DataInputStream 通过提供更多特殊的接口修饰 InputStream。下述代码片段从 URL 对象中直接获取 InputStream 流并加以修饰:

```
URL whiteHouse = newURL("http://www.whitehouse.gov/" );
InputStream undecoratedInput = whiteHouse.openStream( );
DataInputStream decoratedInput = new DataInputStream (undecoratedInput);
```

另一个更灵活的连接远程资源的途径是使用 URL 类的 openConnection( )方法。该方法返回一个 URLConnection 对象,该对象提供了大量的强有力方法,使用这些方法用户可以定制与远程资源的连接。

例如,与 URL 类不同,URLConnection 类允许用户同时获得 InputStream 和 OutputStream 对象。这一点对于实现 HTTP 协议非常有意义。因为该协议中包含 GET 和 POST 访问方法。应用程序只要请求资源就可以使用 GET 方法读取相应的响应信息。而 POST 方法通常通过请求资源的方式对服务器应用程序进行输入,该方法使用 HTTP 请求体将数据写到服务程序并读取响应。为了使用 POST 方法使得用户可以在从 InputStream 中读取数据之前通过 URLConnection 往 OutputStream 中写入。如果用户先执行读取,那么使用 GET 方法,而后续的写入操作将是无效的。

下面的代码片段演示了通过使用 URLConnection 对象并使用 HTTP POST 方法通过写入由 PrintStream 实例修饰 OutputStream 与远程的服务器应用程序连接起来。www.Javasoft.com 利用 CGI 服务器应用程序测试这些方法。这些代码与一个转换 POST 数据的 CGI 应用程序连接并从被修饰过的 InputStream 流中读取被转换过的数据。

```
URL reverseURL = new URL("http://www.javasoft.com/cgi-bin/backwards");
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
URLConnection reverseConn = reverseURL.openConnection( );
PrintStream output = new PrintStream(reverseConn.getOutputStream());
DataInputStream input =
                new DataInputStream(reverseConn.get InputStream());
output.println("String = TexttoReverse");
String reversed = input.readLine( )
```

30.4.3　以 HTTP 为中心的类

在读完关于 URL 类和 URLConnection 类的概述之后，用户可能会认为 URL 类的设计者在开始设计这些类的方法之前就已经对 HTTP(超文本传输协议)协议有了充分的考虑了。当用户看完该类的完整说明之后，用户会对这种看法更为确信，尽管 http 方案只是 Java 众多的 URL 分类中的一种，但是这些类都是以 HTTP 为中心的。

HTTP 可能是用户在网络和国际互联网上进行通信时使用的标准协议，所以用户不必过分担心这方面的限制。但是用户应当清楚地认识到这些方法中的绝大部分都只适用于 HTTP URL。

30.4.4　示例：定制 AltaVista 搜索引擎

既然用户已经了解了 Java 网络部分的基础知识，那么我们就可以编写一些实用程序了。AltaVista 搜索引擎是搜索网页上文档的强有力工具，然而，为了提高效率，它一次只返回几个搜索结果。如果用户开发一个客户应用程序请求所有的搜索结果并将它们完全地显示出来，那将是很有用处的。接下来的 AltaVistaList.java 程序将完成这项工作。

> **提示**：AltaVista 是一个可以为用户在网页上搜索文档的强有力搜索引擎。AltaVista 的主页地址是：http://www.altavista. digital.com/。

在设计这个应用程序时，用户首先要考虑该类应该拥有的公共方法。该类需要包括：

- 一个启动该应用程序的方法
- 一个初始化对象的方法
- 一个打印出所有结果的方法
- 一些被本对象使用的受保护方法：一个用于创建发给 AltaVista 的查询的方法，一个用于从 AltaVista 获取单个网页的方法和一个用于从其它返回的 HTML 页中分析查找结果的方法。

程序清单 30.1 显示了 AltaVistaList 应用程序的完整代码。当用户使用一系列关键字执行该程序时，它返回一个 HTML 页，该页包含了所有由 AltaVista 返回的连接。

> **警告**：该应用程序定制 AltaVista 搜索的结果集合的特殊机制限制。该机制并非一成不变，所以 AltaVista 改变了它，那么本应用程序就不能正常运行了。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 30.1 AltaVistaList.java

```
import java.net.*;    // Import the names of the classes
import java.io.*;     // to be used.

/**
 * This application creates a single, concise HTML page
 * of hits from the AltaVista search engine given a
 * search string.
 * @author David W. Baker
 * @version 1.1
 */
public class AltaVistaList {
   private static final String AGENT_NAME ="Java-alta-search";
   private static final String AGENT_VERSION = "1.0";
   private static final String SEARCH_URL ="http://www.altavista.digital.com/cgi-bin/query";
   private int totalHits = 0;
   private StringBuffer outputList = new StringBuffer();

   /**
    * This starts the application.
    * @param args Program arguments - the search string.
    */
   public static void main(String[] args) {
      if (args.length == 0) {
         System.out.println("Usage: AltaVistaList search string");
         System.exit(1);
      }
      AltaVistaList runApp = new AltaVistaList(args);
      runApp.printOutput(System.out);
      System.exit(0);
   }

   /**
    * This constructor connects to AltaVista and obtains
    * all of the relevant hits.
    * @param args The search tokens.
    */
   public AltaVistaList(String[] args) {
      String hitData;        // Store incoming data.
      int startHits = 0;     // Get the next 10 hits from here.
      String searchSyntax = createQuery(args);

      URLConnection.setDefaultRequestProperty("User-Agent",
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        AGENT_NAME + "/" + AGENT_VERSION);
    while (true) {
        hitData = getPage(SEARCH_URL + "?" + searchSyntax +
            startHits); // Go get a page of hits.
        hitData = getHits(hitData);  // Extract the hits.
        // If there were no hits in the page, hitData will
        // be null. If there were hits, append them to
        // the outputList, increment to the next 10 hits,
        // and go through the loop again.
        if (hitData != null) {
            outputList.append(hitData + "\n");
            startHits += 10;
        // Otherwise, break from the loop.
        } else {
            break;
        }
    }
}

/**
  * This method builds an AltaVista search query
  * string.
  * @param searchTokens An array of search tokens.
  * @return The search query string built.
  */
protected String createQuery(String[] searchTokens) {
    StringBuffer searchString = new StringBuffer();

    // Append the tokens to a single string.
    for(int index = 0; index < searchTokens.length;
            index++) {
        searchString.append(searchTokens[index]);
        // Add a space if there's another token coming up.
        if (index < searchTokens.length-1) {
            searchString.append(" ");
        }
    }
    // URL encode the string.
    String encodedSearchString = URLEncoder.encode(searchString.toString());
    // Return the proper query string.
    return "what=web&fmt=c&pg=q&q=" + encodedSearchString + "&stq=";
}

/**
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    * This method obtains a page from the Web.
    * @param url The URL of the page to obtain.
    * @return The page obtained.
    */
  protected String getPage(String url) {
      // A buffer for the incoming page.
      StringBuffer page = new StringBuffer();
      String nextLine;  // The next line in the input stream.

      try {
          URL urlObject = new URL(url);
          URLConnection agent = urlObject.openConnection();
          DataInputStream input = new DataInputStream(agent.getInputStream());
          // While readLine() doesn't return null, append
          // then next line to the buffer.
          while((nextLine = input.readLine()) != null) {
              page.append(nextLine + "\n");
          }
          input.close();
      } catch(MalformedURLException excpt) {
          System.out.println("Badly formed URL: " + excpt);
      } catch(IOException excpt) {
          System.out.println("Failed I/O: " + excpt);
      }
      // Convert the buffer to a string and return.
      return page.toString();
  }

  /**
    * This method extracts the list of hits from a returned
    * AltaVista results page.
    * @param hitPage The page returned from AltaVista.
    * @return The list of hits.
    */
  protected String getHits(String hitPage) {
      int first,last;       // Begin/end of a substring.
      int notFound = -1;  // Not found return for indexOf().
      String hitSection = null;  // The hits part of page.

      // Go to the first "<a href=" after "<pre>".
      first = hitPage.indexOf("<pre>") + "<pre>".length();
      first = hitPage.indexOf("<a href=",first);
      // End pointer at "</pre>".
      last = hitPage.indexOf("</pre>");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    // If our beginning is after our end, return.
    if (last < first) {
        return hitSection;
    }
    // If neither substring is found, return.
    if (first == notFound || last == notFound) {
        System.err.println("Bad search page format");
        return hitSection;
    }

    // Cut out the substring.
    hitSection = hitPage.substring(first,last);
    first = last = 0;
    totalHits += 1;    // Found one hit.
    // Go through the page line by line.
    while((last = hitSection.indexOf("\n",first))
            != notFound) {
        // Find the next "<a href=" which should be
        // immediately after the \n.
        first = hitSection.indexOf("<a href=",last);
        // If it's not, return the current substring.
        if (first != (last + 1)) {
            return hitSection.substring(0,last);
        // Otherwise, another hit has been found.
        } else {
            totalHits += 1;
        }
    }
    return hitSection; // Return the substring.
}

/**
  * This method prints the list of hits obtained from
  * AltaVista.
  * @param sendOutput Where to print the output.
  */
public void printOutput(PrintStream sendOutput) {
    sendOutput.print("<!DOCTYPE HTML PUBLIC \"-//IETF//" +
        "DTD HTML//EN\"> \n<HTML> \n<HEAD> \n<TITLE>" +
        AGENT_NAME + "</TITLE> \n</HEAD> \n<BODY> \n<H1>" +
        "Search Results</H1> \n<P> <STRONG> Total number of " +
        "hits: " + totalHits + "</STRONG> </P> \n<PRE> \n" +
        outputList + "</PRE> \n</BODY> \n</HTML> \n");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

mian( )方法:启动应用程序 首先,该应用程序引入了两个将要用到的包,引入这两个包可以使得调用 Java 应用编程接口更加简捷。然后声明了该类并初始化了一些私有的实例变量:search _ URL 是 AltaVista 搜索引擎的 URL,totalHits 维护一些返回结果,outputList 是含有返回结果的 HTML 页的缓冲区。

main( )方法允许该代码作为应用程序来执行。该方法检查一套参数并创建 AltaVistList 类的实例。然后由该实例通过向 System.out 传递 PrintOutput( )方法来打印它的输出。System.out 是在 java.lang.System 类中对 PrintStream 对象的引用。通过向 printOutput( )传递 PrintStream 指示 AltaVistaList 实例将它的数据送往标准的输出。最后,main( )方法退出应用程序并返回 0 表示运行正常结束。

AltaVistaList 构造器 该类的唯一一个构造器使用对字符串对象数组的引用作为参数,它将该数组引用传递给 createQuery( )方法,由该方法建立对 AltaVista 搜索引擎的查询。然后由构造器调用 URLConnection 类的静态方法 setDefaultRequestProperty( ):

```
URLConnection.setDefaultRequestProperty("User-Agent", AGENT _ NAME + "/" + AGENT _ VERSION);
```

注意,因为这是一个静态方法,所以不必通过 URLConnection 的实例来调用。该方法可以通过类本身来调用。该方法指示所有的 HTTP 请求应当指定"User-Agent"域作为标识该应用程序的等价字符串。HTTP 的 User-Agent 是一个域,网络客户用它来告诉服务器它们的程序名。以便服务器跟踪客户访问该站点。例如,Netscape 浏览器发送一个包含"Mozilla"的 User-Agent 域,而 Internet Explorer 发送"Explorer"。不幸的是,在编写本章内容时,JDK 还没有实现该方法,User-Agent 仍保留在缺省的 Java < version > 中。该代码被保留下来并假设 JDK 会很快地实现 Java 应用编程接口。

然后该构造器进入一个循环。在这个循环中,它使用一个要接收的网页的 URL 作为参数调用 getPage( )方法。这个 URL 是 AltaVista 搜索引擎的 URL 加上一个问号、由 createQuery( )返回的查询和一个从查询到的结果列表中开始的位置号。AltaVista 每次列出十个查询结果,startHits 计数器让 AltaVistaList 程序根据整个查询结果列表计数。返回页被传递给 getHits( )方法,该方法将查询结果从 HTML 页中抽取出来,如果剩下的结果为空,则从循环中跳出,否则将数据加到 StringBuffer 中并增加计算器的值,为下一次十个查询结果作好准备。

CreateQuery( )方法:建立查询字符串 这是一个受保护的方法,是由 AltaVistaList 类的实例使用的,用于为 AltaVista 搜索引擎建立查询语法。该查询的格式如下:

```
what = web&fmt = c&pg = q&q = < search string > &stq = < n >
```

这一串字符看起来有些让人费解,其实这只是被"&"分隔开的 5 个参数。"what = web"告诉 AltaVista 去搜索它的网页索引;"fmt = c"请求按压缩的格式返回输出以方便后面的分析过程,使得分析和表达更加有效;"pq = q"表明使用基本的语法语言进行简单的查询,"q = "指示用户的搜索字符串,该字符串必须以 URL 格式编码,这里需要对空格和其它特殊字符编码。最后"stq = "告诉 AltaVista 从哪个查询结果开始返回数据,AltaVista 一次返回十个结果,"stq = "允许用户从前十个结果后取得新结果。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

CreateQuery( )有一个数组对象并将整个数组附加到 StringBuffer 对象上，每个字符串都使用空格分隔，另外还用到了 URLEncoder 类中的 encode( )静态方法，这是一个非常有用的方法。

```
String encodedSearchString = URLEncoder.encode(searchString.toString());
```

URLEncoder.encode( )静态方法以一个字符串作为输入参数，它返回一个相应的按 URL 格式编码的字符串，该格式允许将空格和其它特殊字符封装在 URL 中。

createQuery( )接着用一个嵌入的字符串对象返回一个正确的查询。注意，createQuery( )省掉了"stq="参数中的数字，因为该信息已经加入到 main( )方法中了。

getPage( )方法　获取一个网页　getPage( )方法解释了 Java 和网络有关的概念，它接收一个含有 URL 的字符串，然后创建一个 URL 实例并从该实例中获得一个 URLConnect。它设置 DataInputSteam 从连接中逐行地读取数据，然后进入一个循环。该 while 循环读取下一行数据，如果下一行为空则退出该循环。该方法将每一行追加到 StringBuffer 对象上，一旦全部完成则使用该类的 toString( )方法返回一个字符串对象。

getHits( )方法　分析查询结果　该方法的逻辑稍微有些令人费解：它的目的是从 AltaVista 搜索引擎取出一个含 HTML 页的串并将返回的查询结果找出，它通过两个指针来完成这项工作，first 和 last，这两个指针分别指出所分析的页中的子串的头和尾。

getHits 方法查找在 < pre > 第一次出现后的第一个 < a href = 串，这个地方是查询结果的开始处，结束的地方应位于第一个 < /pre > 标识处。如果没有找到上述字符串，该方法返回空的 String 引用，表示本页不含有要搜索的内容。

否则，该方法将扫描整个 HTML 文件，并重复上述过程。每一个查询结果都应是以"< a herf = "开始的新行。该方法将一直扫描下去，直到最后，每遇到这样的新行都表明找到了一个新的查询结果，这时该方法增加用于累计结果树的实例变量的值，并显示当前查找到的结果的总数，在结束时，该方法返回一个包含这些查询结果的字符串。

printOutput( )方法　显示结果　printOutput 方法显示结果非常简单，它以 PrintStream 作为参数，并将 HTML 页打印到一个流中，在本方法中没有使用硬编码的 System.out，printOutput 方法比较灵活，它允许将输出很容易地定向到某些其它的流中。在这个打印出来的 HTML 页中包含查询结果总数和一个格式化的分行的查询结果的部分。

运行 AltaVista　为了运行 Altavista 应用程序，用户首先应使用 javac 将其编译。然后使用 Java 解释器执行它，执行时需要向其传递一套参数。例如，为了看到该网页上附加的信息可以使用 Java 和 URL 作为参数，如下面的命令所示：

```
java AltaVistaList Java URL
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 31 章　TCP 套接字

本章内容：

31.1　TCP 套接字基础
31.2　创建一个 TCP 客户/服务器应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

31.1　TCP 套接字基础

套接字是一个编程抽象概念，它把用户代码与 TCP/IP 协议堆栈的底层实现隔离开了。TCP 套接字可使用户快速地开发出自定制的客户/服务器应用程序。虽然在第 30 章“通信和网络”中所描述的 URL 类非常有用，具有完善的协议，但套接字则允许用户开发出自己的通信模式。

作为一个编程接口，套接字最初是在加利福尼亚大学伯克利分校，作为可以方便地实现网络编程的工具发展起来的。套接字起初是 UNIX 操作系统的一部分，现在它已经广泛地集成到了各种操作环境中，包括 Java。

31.1.1　什么是套接字

套接字是网络上与另一应用程序通信连接的一个句柄。TCP 套接字使用 TCP 协议，并继承了传输协议的行为。创建一个 TCP 套接字需要四方面的信息：

- 本地系统的 IP 地址
- 本地应用程序正在使用的 TCP 端口号
- 远程系统的 IP 地址
- 远程应用程序正在响应的 TCP 端口号

> **提示**：原始的 TCP 规范 RFC 793 中，使用套接字这一术语表示一个系统的 IP 地址与端口号的结合，一对套接字标识一个唯一的端到端的 TCP 连接。在此处的讨论中，套接字使用于一个更高的层次上，一个套接字指的是针对单一网络连接的用户接口。RFC 793 可在以下地址上找到：
>
> ftp://ftp.internic.net/rfc/rfc793.txt

套接字常用于客户/服务器应用程序。一个中心化的服务程序等待各种远程计算机请求特定资源，并且当请求到达时处理每个请求。对客户来讲，要知道如何与服务器通信，标准的应用程序协议分配给了常用端口。在 UNIX 操作系统中，1024 以下的端口只能绑定到具有超级用户(如 root)权限的应用程序，为了便于控制，这些常用的端口放在这个范围内。表 31.1 中列出了一些特定的端口。

> **提示**：Internet 号码分配权威(Internet Assigned Numbers Authority (IANA))给应用程序协议分配了特定的端口。在编写本书时，当前常用端口在 RFC 1700 中列出，从下述地址可以获取：
>
> ftp://ftp.internic.net/rfc/rfc1700.txt

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 31.1 常用 TCP 端口及服务程序

端口	服务程序
21	FIP
23	Telnet
25	SMTP(标准邮件传输协议)
27	Finger
80	HTTP

提示:对许多应用程序来说,用户只能使用 Telnet 应用程序连接于服务端口并手工模拟客户。这可以帮助我们理解客户/服务器的通信方式。

客户应用程序还必须获得一个端口或绑定一个端口,以建立一个套接字连接。由于客户初始化与服务器的通信,在运行时这样的一个端口可以很方便地被分配。客户应用程序通常由一般的非特权用户在 UNIX 系统下运行,所以这些端口是从 1024 以上的范围内分配。当移植到其它系统时,这一规则仍然有效,并且分配给客户应用程序的通常是一个动态分配的或临时的 1024 以上的端口。

由于在同一个机器上不可能将两个应用程序同时绑定于一个端口,所以一个套接字唯一地标识了一个通信连接。但是我们必须明白,一个服务器是可以在同一个端口上响应两个客户的。因为这些客户是在不同的系统或不同的端口,这种连接唯一性的特征仍然存在。图 31.1 说明了这一概念。

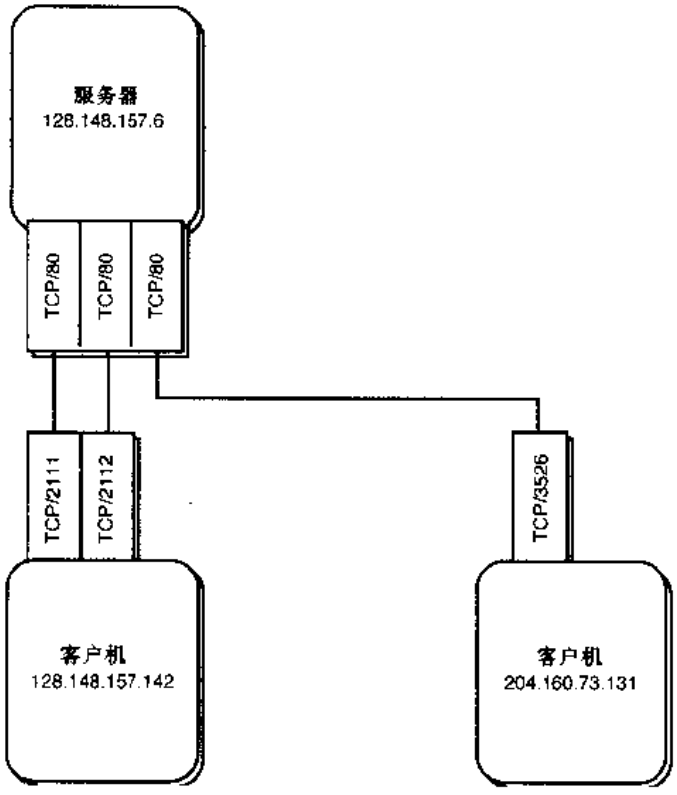


图 31.1 多个客户可通过独立的套接字,连接到一个单一服务器

在图 31.1 中显示了一个服务器应用程序通过 80 端口响应三个套接字的情况,80 端口是

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

HTTP 的常用端口。其中两个套接字正在与同一远程机通信，而另一个套接字则与一个独立的系统通信。注意四个 TCP 套接字的唯一结合性特征。

图 31.1 还显示了一个客户——服务器连接的简化视图。大多数机器使用多重 IP 接口配置——即它们有多个 IP 地址。这些明确的 IP 地址允许保持独立的连接。所以，一个服务器可能有一个应用程序在 80 端口上，正在对一个 IP 地址接收连接，同时另一个应用程序也在 80 端口上，正在对另一个 IP 地址处理连接。这些连接是独立的。在下一节"Java TCP 套接字类"中将介绍的 Java 套接字类，允许用户对连接选择特定的本地接口。

31.1.2　Java TCP 套接字类

Java 有多个类允许用户创建基于套接字的网络应用程序。用户所使用的两个类包括 Java.net.Socket和 Java.net.ServerSocket。

Socket 类用于一般的双向套接字通信，并且有四种常用的构造器：

```
public Socket(String host,int port)throws UnknownHostException,IOException;
public Socket(InetAddress address,int port)throws IOException;
public Socket(String host,int port,InetAddress localAddr,
➡int localPort) throws UnknownHostException,IOException;
public Socket(InetAddress address,int port,InetAddress localAddr,int localPort)
➡throws UnknownHostException,IOException
```

第一个构造器允许用户在一个 String 实例中指定一个远程机的域名并指定远程端口号，创建套接字。第二个构造器可以让用户使用 InetAddress 对象创建套接字。第三个和第四个构造器与前两个类似，不同的是它们允许用户为连接选择本地接口和端口号。如果用户的机器上有多重 IP 地址，可以通过这些构造器选择指定的接口使用。

InetAddress 是一个对象，存储远程系统的 IP 地址。它没有公共的构造器方法，但是有多个返回 InetAddress 实例的静态方法。因此，可通过静态方法调用来创建 InetAddress 对象：

```
try{
InetAddress remoteOP = InetAddress.getByName ("www.microsoft.com");
InetAddress[] allRemoteIPs = InetAddress.getAllByName("www.microsoft.com");
InetAddress.myIP = InetAddress.getLocalHost();
} catch (UnknownHostException excpt){
System.err.println ("Unknown host:" + excpt);
```

提示：在第 30 章中所描述的 DNS 是一个其信息随着时间不断变化的分布式数据库。然而，在 Java 每次运行时 InetAddress 类只对每个主机名执行一次 DNS 解析，所有为特定主机名创建的 InetAddress 后续对象，都将从一个固定的 cache 返回。

这样，如果用户有一个长时间运行的 Java 应用程序，则包含在 InetAddress 对象中的 IP 地址可能会不正确。JDK 代码中的注释说明这是出于安全的考虑。在用户编程时，这是应当注意的一个重要因素。

参见 30.2.1"Internet 协议(IP)"。

Socket 类中包含允许用户通过使用套接字进行读和写的方法，即 getInputStream()和 getOutputStream()方法。为了使应用程序的设计较为容易，这些方法所返回的流通常使用另

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

一个 java.io 对象如 BufferedReader 和 PrintWriter 分别进行了修饰，getInputStream 和 getOutPutStream 均产生一个 IOException 异常，该异常应该可以捕获到。注意下述代码：

```
try {
Socket netspace = new Socket("www.netspace.org",7);
BufferedReader input = new BufferedReader(
new InputStreamReader(netspace.getInputStream()));
PrintWriter output = new PrintWriter(
netspace.getOutputStream(),true);
} catch(UnknownHostException expt) {
System.err.println("Unknown host: " + expt);
System.exit(1);
} catch(IOException expt) {
System.err.println("Failed I/O: " + expt);
System.exit(1);
}
```

为了写一个单行信息并且读一个单行响应，用户仅需使用修饰过的流即可：

```
output.println("test");
String testResponse = input.readLine();
```

当用户通过套接字完成了通信后，必须首先关闭 InputStream 和 OutputStream 实例，然后再关闭套接字。

```
output.close( );
input.close( );
netspace.close( );
```

为了创建一个 TCP 服务器，用户必须首先了解新类 ServerSocket 。ServerSocket 类允许用户绑定一个端口等待客户连接，同时建立一个完整的 Socket 对象。ServerSocket 有三个构造器：

```
public ServerSocket(int port) throws IOException;
public ServerSocket(int port, int count)throws IOException;
public ServerSocket(int port, int count,InetAddress localAddr) throws IOException;
```

第一个构造器在指定的端口创建侦听套接字，允许缺省时的 50 个客户在连接队列中等待。第二个构造器可使用户修改连接队列的长度，允许更多或更少一些客户等待服务器处理。最后一个构造器允许用户指定一个本地接口侦听连接，如果用户的计算机有多个 IP 地址，该构造器允许用户对特定的 IP 地址提供服务。如果用户在这种机器上使用前两个构造器，则 ServerSocket 将接收到与该机器的任意一个 IP 地址的连接。

当 ServerSocket 创建后，可使用 accept()方法等待客户连接。accept()方法在一个客户连接之前一直处于阻塞状态，客户连接后返回一个 Socket 实例，用于与客户通信。阻塞是一个编程术语，意思是一个例程进入了不确定的内部循环，只有当某一特定条件满足时才返回。程序的执行线程在阻塞的例程返回之前，即当特定条件满足之前，不会跳过该例程继续执行。

下述代码在 2222 端口创建了一个 ServerSosket，用来接收一个连接，然后打开流，当客户

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

连接后使用该流进行通信：

```
try {
  ServerSocket server = new ServerSocket(2222);
  Socket clientConn = server.accept();

  BufferedReader input = new BufferedReader(
  new InputStreamReader(clientConn.getInputStream()));
  PrintWriter output = new PrintWriter(
  clientConn.getInputStream(), true);
} catch(IOException excpt) {
  System.err.println("Failed I/O: " + excpt);
  System.exit(1);
}
```

当与该客户通信完毕后，如前所述，服务器必须先关闭流，然后关闭 Socket 实例。

注意：Java API 的套接字类通过使用用户主机的 TCP 实现，提供了方便的流接口。在 JDK 中，抽象类 SocketImpl 的一个子类可以与用户机器的 TCP 进行交互。用户可以定义一个新的 SocketImpl，它使用一个不同于普通 TCP 的传输层。通过创建自己的 SocketImpl 的子类并定义自己的 SocketImplFactory，用户可以修改这种传输层的实现。但在本章中，我们假设用户使用的是 JDK 的套接字工具，它使用的是 TCP。

31.1.3　定制套接字行为

JDK 允许用户指定一些参数，以影响其 TCP 套接字的行为。这些参数模拟 Berkeley 套接字 API 中的某些选项的行为，并且常常通过 API 中的以下几个名字引用：SO_SNDBUF、SO_RCVBUF、SO_TIMEOUT、SO_LINGER 以及 TCP_NODELAY。

Socket 类中的方法 setSoTimeout(int timeout) 允许用户以毫秒为单位指定一个时限，任何随后试图从该套接字的 InputStream 中读取数据的进程，都将等待数据，直到到达该时限。如果到达该时限仍未读取到数据，则将发出一个 InterruptIOException 中断 I/O 异常。缺省情况下，该时限为 0，这意味着一个 read()调用应该永远阻塞或直到一个 IOException 异常发生。

ServerSocket 类也有一个 setSoTimeout(int timeout)方法，但该时限应用于 accept()方法。如果用户设定了一个时限，那么随后调用的 accept()等待一个客户的连接时只等待所指定的毫秒数。当时限到达时，发出 InterruptIOException 异常。对大多数应用程序而言，服务器无限制地等待客户的连接是正确的，但在有些情况下，设置时限会很有用。当异常发出时，ServerSocket 实例仍然有效，并且，如果用户需要，可再次调用 accept()。

Socket 类的 setSoLinger(boolean on, int linger) 方法，允许用户修改 close()方法的行为。通常，当用户使用 close()方法关闭套接字时，排列在我们计算机的 TCP 队列中的数据或已经被接收者所确认的数据，都是在背后处理的；在 TCP 完成传输之前，close()函数并不阻塞。如果我们激活了 setSoLinger()的选项，情况就不同了。当该选项以 0 时限激活时，close()将抛弃掉所有在队列中等待传输的数据，并且向接收者发送一个重置段，即该连接突然中断。当该选项以正数时限激活时，close()将阻塞到所有数据都已发送并且接收者已确认，或直到到达所

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

设置的时限，这种情况下将发出一个 IOException 异常。该选项使我们的应用程序知道，在发送所有数据给接收者时 TCP 的成功程度。

在 JDK1.2 中，Socket 类得到了改进，它允许用户获得并操作发送缓存和接收缓存（SO_SNDBUF 和 SO_RCVBUF）的大小。Socket 的 getSendBufferSize( )和 getReceiveBufferSize( )分别用来获取发送缓存或接收缓存的大小，而 setSendBufferSize(int size)和 setReceiveBufferSize(int size) 则用于指定套接字要素中关于这些缓存所应该有的大小。对于高容量的套接字，缓存设置大一些将提高执行性能，而对于低容量的套接字，或对于那些已经知道块大小的套接字，该数字设置小一些，将减少数据积压。当出现某种错误时，这四种方法都将发出一个 SocketException 异常。

Java TCP 套接字所支持的最后一个套接字选项是 socket 类中 setTcpNoDelay(boolean on)选项中的设置。这将允许用户在其计算机的 TCP 实现中禁用 Nagle 算法。Nagle 算法使得一个 TCP 实现中未确认小节的个数限制为 1。当以小节为单元发送数据时，TCP 使用 Nagle 算法等待每一小节得到确认后，才发送另一小节。这大大减少了网络阻塞，并且在大多数情况下，Nagle 算法的激活对用户是有利的。但是，有时候应用程序必须无延迟地发送少量信息，setTcpNoDelay( ) 方法允许用户禁用这种标准行为。

> **注意**：Nagle 算法最先是在 RFC 896 中提出，可在以下地址中获取：
>
> ftp://ftp.internic.net/rfc/rfc896.txt

31.2 创建 TCP 客户/服务器应用程序

理解了 TCP 套接字编程中的阻塞概念之后，接下来的问题就是开发一个特殊的应用程序。为了说明这一过程，我们将建立一个股票报价服务器和客户机系统。客户将连接到服务器，并请求某组股票的股票信息。服务器将从一个文件中读取数据，定期检查文件是否更新，并将所请求的数据发送给客户。

31.2.1 设计一个应用程序协议

知道了系统的需求后，我们的协议有以下六个基本步骤：

1. 客户连接到服务器。
2. 服务器以一条指示数据当前情况的信息响应客户。
3. 客户请求某一股票的数据。
4. 服务器响应。
5. 重复第 3 和第 4 步，直到客户结束对话。
6. 中止该连接。

为了实现这一设计，我们提出了一个更详细的协议。服务器在端口 1701 上等待客户。当客户首次连接后，服务器以下述方式响应：

```
+ HELLO time-string
```

time-string 指出了所返回的股票数据的最后更新时间。然后，客户发送一条需要信息的请求。服务器随后响应并提供所需数据，如下所示：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
STOCK:stock-id
+ stock-id stock-data
```

stock-id 是一个股票标识符，由一组大写字母组成。stock-data 是一个字符串，详细描述了指定股票的运作情况。通过重复这一请求，客户可以请求关于其它股票的信息。

如果客户发出了一个有关某一股票信息的请求，而服务器中不存在该股票的信息，此时服务器的响应为：

```
-ERR UNKNOWN STOCK ID
```

如果客户发出了一个请求某一股票的命令，但未指出股票 ID，服务器的响应为：

```
-ERR MALFORMED COMMAND
```

如果客户发出了一个无效的命令，服务器的响应为：

```
-ERR UNKNOWN COMMAND
```

当客户完成信息请求后，它将结束通信而服务器要确认对话结束：

```
QUIT
+ BYE
```

下面的例子说明了一个使用上述协议的对话过程。所有服务器的响应都应以一个 + 号或 - 号开始，而客户请求则不以 + 或 - 号开始。在这个例子中，客户正在请求三种股票的信息：ABC、XYZ、及 AAM。服务器中只包含后两种股票的信息：

```
+ HELLO Tue,Jul 16,1996 09:15:13 PDT
STOCK:ABC
-ERR UNKNOWN STOCK ID
STOCK:XYZ
+ XYZ Last: 20 7/8; Change -0 1/4; Volume 60,400
STOCK:AAM
+ AAM Last 35; Change 0; Volume 2,500
QUIT
+ BYE
```

31.2.2　开发股票客户程序

实现上述协议的客户应用程序非常简单，其代码在程序清单 31.1 中给出。

程序清单 31.1　StockQuoteClient.java

```
import java.io.*;      // Import the names of the packages
import java.net.*;     // to be used.

/**
 * This is an application which obtains stock information
 * using our new application protocol.
 * @author David W. Baker
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
 * @version 1.2
 */
public class StockQuoteClient {
    // The Stock Quote server listens at this port.
    private static final int SERVER_PORT = 1701;
    // Should your quoteSend PrintWriter autoflush?
    private static final boolean AUTOFLUSH = true;
    private String serverName;
    private Socket quoteSocket = null;
    private BufferedReader quoteReceive = null;
    private PrintWriter quoteSend = null;
    private String[] stockIDs;      // Array of requested IDs.
    private String[] stockInfo;     // Array of returned data.
    private String currentAsOf = null;  // Timestamp of data.

    /**
     * Start the application running, first checking the
     * arguments, then instantiating a StockQuoteClient, and
     * finally telling the instance to print out its data.
     * @param args Arguments which should be <server> <stock ids>
     */
    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Usage: StockQuoteClient <server> <stock ids> *");
            System.exit(1);
        }
        StockQuoteClient client = new StockQuoteClient(args);
        client.printQuotes(System.out);
        System.exit(0);
    }

    /**
     * This constructor manages the retrieval of the
     * stock information.
     * @param args The server followed by the stock IDs.
     */
    public StockQuoteClient(String[] args) {
        String serverInfo;

        // Server name is the first argument.
        serverName = args[0];
        // Create arrays as long as arguments - 1.
        stockIDs = new String[args.length - 1];
        stockInfo = new String[args.length - 1];
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        // Copy the rest of the elements of the args array
        // into the stockIDs array.
        for (int index = 1; index < args.length; index++) {
            stockIDs[index - 1] = args[index];
        }
        // Contact the server and return the HELLO message.
        serverInfo = contactServer();
        // Parse out the timestamp, which is everything after
        // the first space.
        if (serverInfo != null) {
            currentAsOf = serverInfo.substring(
                              serverInfo.indexOf(" ") - 1);
        }
        getQuotes();    // Go get the quotes.
        quitServer();   // Close the communication.
    }

    /**
     * Open the initial connection to the server.
     * @return The initial connection response.
     */
    protected String contactServer() {
        String serverWelcome = null;

        try {
            // Open a socket to the server.
            quoteSocket = new Socket(serverName, SERVER_PORT);
            // Obtain decorated I/O streams.
            quoteReceive = new BufferedReader(new InputStreamReader(
                               quoteSocket.getInputStream()));
            quoteSend = new PrintWriter(quoteSocket.getOutputStream(), AUTOFLUSH);
            // Read the HELLO message.
            serverWelcome = quoteReceive.readLine();
        } catch (UnknownHostException excpt) {
            System.err.println("Unknown host " + serverName + ": " + excpt);
        } catch (IOException excpt) {
            System.err.println("Failed I/O to " + serverName + ": " + excpt);
        }
        return serverWelcome;    // Return the HELLO message.
    }

    /**
     * This method asks for all of the stock info.
     */
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
protected void getQuotes() {
    String response;  // Hold the response to stock query.

    // If the connection is still up.
    if (connectOK()) {
        try {
            // Iterate through all of the stocks.
            for (int index = 0; index < stockIDs.length; index++) {
                // Send query.
                quoteSend.println("STOCK: " + stockIDs[index]);
                // Read response.
                response = quoteReceive.readLine();
                // Parse out data.
                stockInfo[index] = response.substring(response.indexOf(" ") + 1);
            }
        } catch (IOException excpt) {
            System.err.println("Failed I/O to " + serverName + ": " + excpt);
        }
    }
}

/**
 * This method disconnects from the server.
 * @return The final message from the server.
 */
protected String quitServer() {
    String serverBye = null;   // BYE message.

    try {
        // If the connection is up, send a QUIT message
        // and receive the BYE response.
        if (connectOK()) {
            quoteSend.println("QUIT");
            serverBye = quoteReceive.readLine();
        }
        // Close the streams and the socket if the
        // references are not null.
        if (quoteSend != null) quoteSend.close();
        if (quoteReceive != null) quoteReceive.close();
        if (quoteSocket != null) quoteSocket.close();
    } catch (IOException excpt) {
        System.err.println("Failed I/O to server " + serverName + ": " + excpt);
    }
    return serverBye; // The BYE message.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }

    /**
     * This method prints out a report on the various
     * requested stocks.
     * @param sendOutput Where to send output.
     */
    public void printQuotes(PrintStream sendOutput) {
        // Provided that you actually received a HELLO message:
        if (currentAsOf != null) {
            sendOutput.print("INFORMATION ON REQUESTED QUOTES"
                + "\n\tCurrent As Of: " + currentAsOf + "\n\n");
            // Iterate through the array of stocks.
            for (int index = 0; index < stockIDs.length; index++) {
                sendOutput.print(stockIDs[index] + ":");
                if (stockInfo[index] != null)
                    sendOutput.println("" + stockInfo[index]);
                else sendOutput.println();
            }
        }
    }

    /**
     * Conveniently determine if the socket and streams are
     * not null.
     * @return If the connection is OK.
     */
    protected boolean connectOK() {
        return (quoteSend != null && quoteReceive != null &&
                quoteSocket != null);
    }
}
```

启动客户端的 main()方法　main()方法首先检查应用程序是否以适当的命令行参数激活，如果不正确，退出程序。如果正确，则将以 args 数组引用将 StockQuoteClient 实例化，并运行 PrintQuotes()方法，通知客户向标准输出发送其数据。

StockQuoteClient 构造器　该构造器的目的是初始化数据结构、连接到服务器、从服务器上加载股票数据、以及中止该连接。该构造器创建了两个数组，其中一个用来复制股票的 ID，另一个则未初始化，用来存放各股票的数据。

该构造器使用 contactServer()方法打开与服务的通信，返回一个表示已打开的字符串。如果连接被正确打开，该字符串中包含有一个时间戳，指明股票数据的当前情况。构造器将分析这个字符串，分离出其中的时间戳，使用 getQuotes() 方法获取股票数据，最后使用 quit-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Server()关闭连接。

启动通信的 contactServer() 方法 正如在本章前面的例子中所看到的那样，该方法打开一个到服务器的套接字，然后创建了两个流，用于与服务器进行通信。最后，该方法从服务器接收表示已打开的行(例如，+HELLO time-string)，并以一个 String 将其返回。

获取股票数据的 getQuotes()方法 该方法对用来激活应用程序的每个股票 ID 执行查询，这些股票 ID 目前存放在 stockIDs 数组中。首先，该方法调用了一个很短的方法 connectOK()，它只是用来确认套接字及流为非空，然后反复使用 stockIDs 数组，将每个元素的请求发送到服务器。它还读取每个响应，并从返回行中分析出股票数据。该方法将股票数据作为分散的元素存储在 stockInfo 数组中。当请求每个股票的信息完成后，getQuotes()方法返回。

结束连接的 quitServer()方法 该方法结束与服务器的连接。如果连接仍有效，首先发送一个 QUIT 信息，然后执行结束一个套接字通信时的基本步骤：先关闭流再关闭套接字。

显示股票报价的 printQuotes()方法 给定一个 PrintStream 对象如 System.out，该方法打印股票数据。它通过股票标识符数组 stockIDs 的迭代，打印出相应的 stockInfo 数组中的值。

31.2.3 开发股票报价服务器程序

服务器应用程序与请求其服务的客户程序比较，要稍复杂一些。服务器应用程序实际上由两个类组成，第一个类装载股票数据并等待客户连接。当一个客户实际连接后，它创建另外一个用来实现可运行接口的类的一个实例，并传送新创建的套接字给客户端。

第二个对象是一个句柄，运行于其自身的执行线程。这一点使服务器可以循环起来接收更多的客户，而并非一次只执行与一个客户的通信。对服务器一个接着一个地处理请求，我们称之为迭代；对同时可以处理多个请求的服务器应用程序，我们称之为并发服务器应用程序。对于通常可以持续长时间交互的 TCP 客户/服务器，并发操作是很必要的。句柄是执行与客户实际通信的对象，句柄的多个实例允许服务器同时处理多个请求。

这是一个普通的网络服务器程序设计——它使用了多线程服务器，这使得许多客户的连接可以同时得到处理。该应用程序的代码在程序清单 31.2 中给出。

程序清单 31.2 StockQuoteServer.java

```
import java.io.*;      // Import the package names to be
import java.net.*;     // used by this application.
import java.util.*;

/**
 * This is an application that implements our stock
 * quote application protocol to provide stock quotes.
 * @author David W. Baker
 * @version 1.2
 */
public class StockQuoteServer {
   // The port on which the server should listen.
   private static final int SERVER_PORT = 1701;
   // Queue length of incoming connections.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
private static final int MAX_CLIENTS = 50;
// File that contains the stock data of format:
// <stock-id> <stock information>
private static final File STOCK_QUOTES_FILE = new File("Stockquotes.txt");
private ServerSocket listenSocket = null;
private Hashtable stockInfo;
private Date stockInfoTime;
private long stockFileMod;
// A boolean used to keep the server looping until
// interrupted.
private boolean keepRunning = true;

/**
 * Starts up the application.
 * @param args Ignored command line arguments.
 */
public static void main(String[] args) {
    StockQuoteServer server = new StockQuoteServer();
    server.serveQuotes();
}

/**
 * The constructor creates an instance of this class,
 * loads the stock data, and then our server listens
 * for incoming clients.
 */
public StockQuoteServer() {
    // Load the quotes and exit if it is unable to do so.
    if (! loadQuotes()) System.exit(1);
    try {
        // Create a listening socket.
        listenSocket = new ServerSocket(SERVER_PORT,MAX_CLIENTS);
    } catch(IOException excpt) {
        System.err.println("Unable to listen on port " + SERVER_PORT + ": " + excpt);
        System.exit(1);
    }
}

/**
 * This method loads in the stock data from a file.
 */
protected boolean loadQuotes() {
    String fileLine;
    StringTokenizer tokenize;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
String id;
StringBuffer value;

try {
   // Create a decorated stream to the data file.
   BufferedReader stockInput = new BufferedReader(
       new FileReader(STOCK_QUOTES_FILE));
   // Create the Hashtable in which to place the data.
   stockInfo = new Hashtable();
   // Read in each line.
   while ((fileLine = stockInput.readLine()) != null) {
     // Break up the line into tokens.
     tokenize = new StringTokenizer(fileLine);
     try {
       id = tokenize.nextToken();
       // Ensure the stock ID is stored in upper case.
       id = id.toUpperCase();
       // Now create a buffer to place the stock value in.
       value = new StringBuffer();
       // Loop through all remaining tokens, placing them
       // into the buffer.
       while(tokenize.hasMoreTokens()) {
         value.append(tokenize.nextToken());
         // If there are more tokens to come, then append
         // a space.
         if (tokenize.hasMoreTokens()) {
           value.append(" ");
         }
       }
       // Create an entry in our Hashtable.
       stockInfo.put(id, value.toString());
     } catch(NullPointerException excpt) {
       System.err.println("Error creating stock data " +"entry: " + excpt);
     } catch(NoSuchElementException excpt) {
       System.err.println("Invalid stock data record " +"in file: " + excpt);
     }
   }
   stockInput.close();
   // Store the last modified timestamp.
   stockFileMod = STOCK_QUOTES_FILE.lastModified();
} catch(FileNotFoundException excpt) {
   System.err.println("Unable to find file: " + excpt);
   return false;
} catch(IOException excpt) {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            System.err.println("Failed I/O: " + excpt);
            return false;
        }
        stockInfoTime = new Date();    // Store the time loaded.
        return true;
    }

    /**
     * This method waits to accept incoming client
     * connections.
     */
    public void serveQuotes() {
        Socket clientSocket = null;

        try {
            while(keepRunning) {
                // Accept a new client.
                clientSocket = listenSocket.accept();
                // Ensure that the data file hasn't changed; if
                // so, reload it.
                if (stockFileMod != 
                    STOCK_QUOTES_FILE.lastModified()) {
                    loadQuotes();
                }
                // Create a new handler.
                StockQuoteHandler newHandler = new
                    StockQuoteHandler(clientSocket,stockInfo,stockInfoTime);
                Thread newHandlerThread = new Thread(newHandler);
                newHandlerThread.start();
            }
            listenSocket.close();
        } catch(IOException excpt) {
            System.err.println("Failed I/O: "+ excpt);
        }
    }
    /**
     * This method allows the server to be stopped.
     */
    protected void stop() {
        if (keepRunning) {
            keepRunning = false;
        }
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
/**
 * This class is used to manage a connection to
 * a specific client.
 */
class StockQuoteHandler implements Runnable {
    private static final boolean AUTOFLUSH = true;
    private Socket mySocket = null;
    private PrintWriter clientSend = null;
    private BufferedReader clientReceive = null;
    private Hashtable stockInfo;
    private Date stockInfoTime;

    /**
     * The constructor sets up the necessary instance
     * variables.
     * @param newSocket Socket to the incoming client.
     * @param info The stock data.
     * @param time The time when the data was loaded.
     */
    public StockQuoteHandler(Socket newSocket,
                             Hashtable info, Date time) {
        mySocket = newSocket;
        stockInfo = info;
        stockInfoTime = time;
    }

    /**
     * This is the thread of execution that implements
     * the communication.
     */
    public void run() {
        String nextLine;
        StringTokenizer tokens;
        String command;
        String quoteID;
        String quoteResponse;

        try {
            clientSend =
                new PrintWriter(mySocket.getOutputStream(),AUTOFLUSH);
            clientReceive = new BufferedReader(new InputStreamReader(mySocket.getInputStream()));
            clientSend.println("+ HELLO "+ stockInfoTime);
            // Read in a line from the client and respond.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        while((nextLine = clientReceive.readLine())
              != null) {
            // Break the line into tokens.
            tokens = new StringTokenizer(nextLine);
            try {
              command = tokens.nextToken();
              // QUIT command.
              if (command.equalsIgnoreCase("QUIT")) break;
              // STOCK command.
              else if (command.equalsIgnoreCase("STOCK:")) {
                 quoteID = tokens.nextToken();
                 quoteResponse = getQuote(quoteID);
                 clientSend.println(quoteResponse);
              }
              // Unknown command.
              else {
                 clientSend.println("- ERR UNKNOWN COMMAND");
              }
            } catch(NoSuchElementException excpt) {
              clientSend.println("-ERR MALFORMED COMMAND");
            }
        }
        clientSend.println(" + BYE");
    } catch(IOException excpt) {
        System.err.println("Failed I/O: " + excpt);
    // Finally close the streams and socket.
    } finally {
        try {
            if (clientSend != null) clientSend.close();
            if (clientReceive != null) clientReceive.close();
            if (mySocket != null) mySocket.close();
        } catch(IOException excpt) {
            System.err.println("Failed I/O: " + excpt);
        }
    }
}

/**
  * This method matches a stock ID to relevant information.
  * @param quoteID The stock ID to look up.
  * @return The releveant data.
  */
protected String getQuote(String quoteID) {
    String info;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        // Make sure the quote ID is in upper case.
        quoteID = quoteID.toUpperCase();
        // Try to retrieve from out Hashtable.
        info = (String)stockInfo.get(quoteID);
        // If there was such a key in the Hashtable, info will
        // not be null.
        if (info != null) {
          return "+" + quoteID + "" + info;
        }
        else {
          // Otherwise, this is an unknown ID.
          return "-ERR UNKNOWN STOCK ID";
        }
    }
}
```

启动服务器程序 main()方法允许该服务器程序像一个应用程序一样启动,并实例化一个新的 StockQuoteServer 对象,然后使用 serveQuotes() 方法开始接收客户连接。

构造器首先调用 loadQuotes()方法加载股票数据。构造器确保该过程能成功,如果不成功,退出应用程序。另外,它在端口 1701 创建了一个 ServerSocket。现在服务器程序正在等待客户进入。

读入股票数据的 loadQuotes( )方法 该方法使用了一个 java.io.File 对象,用来获取一个 DataInputStream 流,从"stockquotes.txt"数据文件中读入数据。loadQuotes()扫描文件的每一行,寻找每一个以下述格式表示的新的股票对应的行:

```
Stock-ID stock-data
```

该方法分析本行的信息,将数据置于散列表(Hashtable)实例中;股票 ID 的大写字母是其键字,而股票数据是其值。该方法使用 File 类的 lastModified()方法存储该文件的修改时间,以便当数据更新后服务器能够检测到。它使用 java.util.Date 类存放当前日期,以便它能够告诉连接客户股票信息装载的时间。

在更理想的设计中,该方法应该是从股票信息的实际源读取数据。由于我们可能尚未与别的公司建立起这样的服务,所以我们现在采用静态文件。

响应进入的客户的 serveQuotes()方法 该方法在一个无限循环中运行,当客户进入时建立与客户的连接。该方法在 ServerSocket 的 accept()方法处阻塞,等待客户连接。当连接建立后,该方法检查存放股票数据的文件,是否自上次加载后进行过修改,如果是,它将调用 loadQuotes 方法重新加载数据。

然后,serveQuotes()方法将创建一个 StockQuoteHandler 实例,将客户连接时创建的套接字和股票数据列表传送给它。它将该句柄置于一个线程对象内,并启动该线程的执行。当这一过程执行完后,serveQuotes()方法循环到开始,等待新的客户连接。

创建 StockQuotesHandler 该类实现了一个可运行接口,以便使它能在自己的线程中运

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

行。构造器仅仅设置了一些实例变量，用来引用套接字及传送给它的股票数据。

实现通信的 run()方法　该方法打开两个流，用于从客户端读和向客户端写数据。它首先向客户发送连接已打开信息，然后从客户读取每一个请求。该方法使用一个 StringTokenizer 来分析客户的请求，并将其与它所支持的两个命令：STOCK:和 QUIT 进行匹配。

如果请示是一个 STOCK:命令，它将假设 STOCK:之后的标记为股票标识符，并将该标识符传送给 getQuote()方法，以获得合适的数据。getQuote()是一个简单方法，用来在 stockInfo 散列表中查找匹配行。如果找到了匹配行，则将其返回，否则，将返回一个错误信息。run()方法将向客户端发送这些信息。

如果请示是一个 QUIT 命令，服务器发送 + BYE 响应，并中断循环。然后通过关闭流和套接字结束通信。此时，run()方法结束，并允许运行该对象的线程终止。

如果请示不是这两个命令中的任何一种，服务器发送回一个错误信息，并等待客户的合法命令请求。

31.2.4　运行客户和服务器应用程序

用 javac 编译这两个应用程序。然后检查是否已建立了服务器程序代码中所指定的、有着适当格式的股票报价数据文件 stockquotes.txt。使用 Java 解释器运行服务程序，它将一直运行到系统中断它为止。

最后，运行客户端程序，查看服务器如何响应。再试着使用一个或多个存放于数据文件中的股票标识符运行客户程序。最后，更新数据文件并再次进行查询，客户端应显示出数据已经修改的提示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 32 章　UDP 套接字

本章内容：

32.1　UDP 信息概览
32.2　创建 UDP 服务器程序
32.3　创建 UDP 客户程序
32.4　使用 IP 多点发送

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

32.1　UDP 信息概览

对大多数 Internet 开发者来说，使用 UDP(用户数据报协议，User Datagram Protocol)的情况要比使用 TCP 少得多。UDP 并不完全将用户从实现连续网络通信的细节中分离出来。但是，对大多数 Java 应用程序而言，选择 UDP 作为创建网络连接的工具，也许是其最谨慎的选择。

使用 UDP 编程时存在很多支节问题，理解这些因素将会对用户的网络编程起到指导和帮助作用。

UDP 对于那些可以将通信分解成离散信息块的应用程序是一种很好的选择。在这种情况下，来自客户的单一查询激活服务器的一个响应。与时间有关的数据尤其适合于 UDP。UDP 需要的系统开销较少，但是任何对于系统所必须的可靠性的工程就都成了开发者的责任。例如，如果客户的查询从未收到响应——使用 UDP 时这是完全可能并且合法的——很可能需要客户程序重新发送请求或者显示一条提示信息以表示出现通信故障。

32.1.1　UDP 套接字的特性

正如在第 30 章“通信和网络”中所讨论的那样，UDP 的行为与 TCP 有很大的不同。UDP 是不可靠的、无连接的和面向报文的。所以通常将 UDP 类比解释为明信片通信。

UDP 对话必须被量化分解为小的信息块，以使其可以装入到指定尺寸的小数据包中，虽然有些包可包含比其它包更多的数据。当用户发送出一个信息时，他们从来都不能确定是否能收到返回信息。除非用户确实收到了一个返回信息，否则他们无法知道其信息是否被收到——这可能是用户的信息在途中丢失，或者是收信者的确认信息丢失，也可能是接收者忽视了对方的信息。

网络程序之间所交换的这种明信片称为数据报。在一个数据报中，用户可以存放字节数组，接收应用程序可以解开这种数组，并对数据报进行解码，也可能发送一个响应数据报。

与 TCP 一样，UDP 应用程序也使用套接字编程概念，但 UDP 套接字与 TCP 套接字有很大的差别。作为明信片比拟的扩展，UDP 套接字非常类似于创建一个邮箱。

一个邮件信箱是由用户的地址标识的，但用户没有对每个需要发送信息的人都建立一个新的邮箱。(但是，用户可以创建一个新的邮箱来接收报纸，而不是将报纸放入普通信箱中。)相反，是在明信片上写上收件人的地址，把明信片放到邮箱中，并最终得以发送。

在接收报文时，用户可能在不知不觉中永远等待直到有一个报文到达自己的信箱。当报文到达之后，用户可以阅读明信片。出现于明信片上的元信息使用返回地址标识出发送者。

从以上的类比中可以看出，UDP 编程一般包括以下几个任务：

- 创建一个发往合适地址的数据报
- 为一特别的应用程序设置一个套接字，用于发送和接收数据报
- 在传送套接字中插入数据报
- 等待从一套接字接收数据报
- 对数据报进行解码，提取报文、接收者及其它元信息

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

32.1.2 Java UDP 类

在 Java.net 包中有执行 UDP 通信所必须的工具。为了建立数据报，Java 提供了 DatagramPacket 类。当接收一个 UDP 数据报时，用户也使用 DatagramPacket 类，用于读取数据、发送者及其它元信息。

为创建一个发送到远程系统的数据报，提供了如下的构造器：

```
public DatagramPacket(byte[] ibuf,int length,InetAddress iaddr,int iport);
```

ibuf 是一个字节数组，用于对报文数据进行编码，length 是放进数据报中的字节数组的长度，该值决定了数据报的大小。iaddr 是一个 InetAddress 对象，用来存放接收者的 IP 地址，iport 标识接收主机上接收报文的端口号。

参见 3.1.1.2“Java TCP 套接字类”。

为了接收数据报，用户还必须使用另外一个 DatagramPacket 构造器，用来存放传送来的数据，该构造器的原型为：

```
public DatagramPacket(byte[] ibuf,int ilength);
```

ibuf 是一个用来复制数据报中数据部分的字节数组，ilength 是从数据报复制到与数据大小相应数组中的字节数，如果 ilength 小于机器所接收的 UDP 数据报大小，Java 将自动忽略多余的字节。

> **注意**：使用 TCP 套接字编程时，用户不必将自己的数据分解成离散的数据块，再在网络上传输。当创建一个基于 UDP 的客户/服务器协议时，用户必须指定一些所需的数据报长度，或创建一个在运行时决定这些长度的方法。
>
> 根据 TCP/IP 规范，最大的数据报可包含 65507 个字节的数据，但一个主机只要求接收最多 548 个字节数据的数据报。大多数平台支持更大一些的数据报，即至少 8192 字节长。
>
> 大数据报在 IP 层很可能被分成碎块。如果在传输过程中有包含数据报碎块的 IP 包丢失，则整个 UDP 数据报都将会丢失。关键是在设计应用程序时要注意数据报的长度，要谨慎地将该长度限制为一个合理的长度。

正如本节稍后所讲述的那样，当一个数据报被接收后，用户可以读取数据。其它的方法允许用户获取关于报文的元信息：

```
public int getLength();
public byte[] getData();
public InetAddress getAddress();
public int getPort();
```

getLength()方法用于获取数据报的数据部分所包含的字节数，getData()方法用于获取包含所接收数据的字节数组，getAddress()提供了一个 InetAddress 对象，用来标识发送者，而 getPort()指出所使用的 UDP 端口。

执行发送与接收数据的任务是使用 DatagramSocket 类来完成的，它创建一个 UDP 套接字，共有三种构造器可使用：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public DatagramSocket() throws IOException;
public DatagramSocket(int port) throws IOException;
public DatagramSocket(int port, InetAddress localAddr)throws IOException;
```

第一个构造器允许用户在一个不使用的临时端口上创建一个套接字，一般用于客户应用程序。第二个构造器允许用户指定一个特定端口，它对服务器应用程序非常有用。与使用 TCP 一样，大部分系统对于绑定 1024 以下的端口需要超级用户权限。最后一个构造器用于具有多重 IP 接口的机器，用户可使用该构造器从分配给该机器的 IP 地址之一发送和侦听数据报。在这样的一个主机上，发送到该机器任何 IP 地址的数据报都是由前两个构造器所创建的 DatagramSocket 接收的，而最后一个构造器只获取发送到指定 IP 地址的数据报。

用户可使用该套接字发送具有适当地址的 DatagramPocket 实例，该实例是由第一个构造器创建的，发送所使用的 DatagramSocket 方法为：

```
public void send(DatagramPacket p) throws IOException;
```

当使用第二个构造器描述创建一个 DatagramPacket 后，数据报可由如下方法接收：

```
public synchronized void receive(DatagramPacket  p)throws IOException;
```

注意，receive()方法在收到数据报之前处于阻塞。由于 UDP 是不可靠的，用户的应用程序不能期望 receive()能有返回，除非设置了时限。时限在伯克利套接字 API 选项中的选项名为 SO_TIMEOUT，可由 DatagramSocket 类的下述方法来设置：

```
public synchronized void setSoTimeout(int timeout)
  throws SocketException;
```

timeout 是一个以毫秒为单位的值，如果设置为 0，receive() 方法将使用一个无穷大时限——这是缺省设置。当设置大于 0 时，随后的 receive() 方法将在一个 InterruptedIOException 异常发出之前，只等待指定时限的时间。

注意：用户主机的 UDP 实现对所流入的数据报队列是有限制的。如果用户应用程序不能尽快处理这些数据，它们将悄悄地被抛弃，当数据报从一个队列溢出从而丢失时，不论是发送者还是接收者都是不会察觉到的，这就是 UDP 不可靠的本质。

当使用 UDP 套接字的通信完成后，应当关闭套接字：

```
public synchronized void close();
```

32.2　创建 UDP 服务器程序

本节中我们将学习如何创建基本的 UDP 服务器程序，以响应简单的客户请求。此处所使用的实际例子是创建一个日期服务器程序。

日期服务程序是运行于许多系统的一个简单服务。例如，大多数 UNIX 系统从 inetd 运行日期服务，正如在/etc/inetd.conf 所列出的那样。在 Windows NT 中，日期服务程序可通过 Services Control Panel 中的 Simple TCP/IP Services 获取。日期服务程序一般运行于 UDP 端口 13。当发送一个数据报时，它使用一个包含日期的数据报响应，格式为：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Friday, July 30, 1993 19:25:00

程序清单 32.1 给出了用于实现这种服务的 Java 代码。

程序清单 32.1 DaytimeServer.java

```
import java.net.*;    // Import the package names used
import java.util.*;
import java.io.*;
import java.text.*;

/**
  * This is an application that runs the
  * daytime service.
*/
public class DaytimeServer {
    // The daytime service runs on this well known port.
    private static final int TIME_PORT = 13;
    private DatagramSocket timeSocket = null;
    private static final int SMALL_ARRAY = 1;
    private static final int TIME_ARRAY = 100;
    // A boolean to keep the server looping until stopped.
    private boolean keepRunning = true;

    /**
* This method starts the application, creating an
     * instance and telling it to start accepting
     * requests.
     * @param args Command line arguments - ignored.
     */
  public static void main(String[] args) {
      DaytimeServer server = new DaytimeServer();
      server.startServing();
    }

    /**
     * This constructor creates a datagram socket to
     * listen on.
     */
  public DaytimeServer() {
      try {
        timeSocket = new DatagramSocket(TIME_PORT);
      } catch(SocketException except) {
        System.err.println("Unable to open socket: " + except);
      }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
  }

/**
   * This method does all of the work of listening for
   * and responding to clients.
   */
public void startServing() {
   DatagramPacket datagram;      // For a UDP datagram.
   InetAddress clientAddr;       // Address of the client.
   int clientPort;               // Port of the client.
   byte[] dataBuffer;            // To construct a datagram.
   String timeString;            // The time as a string.

   // Keep looping while you have a socket.
   while(keepRunning) {
     try {
         // Create a DatagramPacket to receive query.
         dataBuffer = new byte[SMALL_ARRAY];
         datagram = new DatagramPacket(dataBuffer,dataBuffer.length);
         timeSocket.receive(datagram);
         // Get the meta-info on the client.
         clientAddr = datagram.getAddress();
         clientPort = datagram.getPort();
         // Place the time into byte array.
         dataBuffer = getTimeBuffer();
         // Create and send the datagram.
         datagram = new DatagramPacket(dataBuffer,dataBuffer.length,clientAddr,clientPort);
         timeSocket.send(datagram);
       } catch(IOException excpt) {
         System.err.println("Failed I/O:" + excpt);
       }
     }
     timeSocket.close();
   }

   /**
    * This method is used to create a byte array
    * containing the current time in the special daytime
    * server format.
    * @return The byte array with the time.
    */
 protected byte[] getTimeBuffer() {
     String timeString;
     SimpleDateFormat daytimeFormat;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        Date currentTime;

        // Get the current time.
        currentTime = new Date();
        // Create a SimpleDateFormat object with the time
        // pattern specified.
        //   EEEE   - print out complete text for day
        //   MMMM - print out complete text of month
        //   dd     - print out the day in month in two digits
        //   yyyy   - print out the year in four digits
        //   HH     - print out the hour in the day, from 0 - 23
        //            in two digits
        //   mm     - print out the minutes in the hour in two
        //            digits
        //   ss     - print out the seconds in the minute in
        //            two digits
        daytimeFormat =
          new SimpleDateFormat("EEEE, MMMM dd, yyyy HH:mm:ss*);

        // Create the special time format.
        timeString = daytimeFormat.format(currentTime);
        // Convert the String to an array of bytes using the
        // platform's default character encoding.
        return timeString.getBytes();
    }

    /**
     * This method provides an interface to stopping
     * the server.
     */
    protected void stop() {
        if (keepRunning) {
            keepRunning = false;
        }
    }

    /**
     * Just in case, do some cleanup.
     */
    public void finalize() {
        if (timeSocket != null) {
            timeSocket.close();
        }
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

32.2.1　启动服务器

在 DaytimeServer 类中，使用了许多静态最终变量作为常量，其中大多数用于以适当格式创建日期串。main()方法创建一个 DaytimeServer 对象，然后激活 startServing()方法，以便接收流入的请求。

DaytimeServer 构造器仅仅在指定的端口创建了一个 UDP 套接字。注意，正如上面所讲的那样，该服务器可能需要超级用户权限来运行，因为它绑定到了端口 13。如果不允许绑定到该端口，试图创建 DatagramSocket 时将发出一个异常，构造器捕获到这一异常，体面地失败，并通知用户问题的所在。

DaytimeServer 是一个迭代式服务器程序，而在第 31 章“TCP 套接字”中创建的服务器程序是一个并发式服务器程序。DaytimeServer 按照请求的到达顺序串行处理每一个请求。对这种给定协议的本质——数据报到达后，服务器立即返回一个数据报——迭代服务器是最适合的，其编程也较为简单。

32.2.2　StartServing()方法处理请求

StartServing()方法实现逻辑服务。应用程序运行时其循环通过如下几步：它创建一个小的字节数组并使用该数组创建一个 DatagramPacket；然后应用程序从 DatagramsSocket 接收一个数据报；从该数据报中获取请求应用程序的 IP 地址和端口。StartServing 方法不需要从流入的数据报中读取任何信息，因为数据报的到达及其所包含的元信息对于服务器理解请求已是足够了。

getTimeBuffer()方法的调用是为了获取包含适当格式的时间字节数组。通过使用这些信息，该方法创建一个新的 DatagramPacket，最后，通过 DatagramSocket 发送这一信息，服务器以这一过程循环，直到发生外部中断。

32.2.3　getTimeBuffer()方法创建字节数组

该保护性方法创建一个包含当前时间的 Date 类实例，并以指定的时间模式实例化一个 simpleDateFormat。它使用 simpleDateFormat 对象，以适当的数据格式创建一个 String，最后，返回与该 String 相对应的字节数组。

32.2.4　运行时间服务器程序

为运行该服务器程序，首先使用 javac 对其进行编译。然后，如果需要，以超级用户（例如“root”）注册，并使用 java 运行服务器程序。如果不需要则修改 TIME_PORT 变量，使其绑定到 1024 以上的端口。

在下面的例子中，我们创建一个连接到该服务器的客户程序。

32.3　创建 UDP 客户程序

该例子用于创建一个使用上述日期时间服务器程序的客户程序，也用来说明通过一个单

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

·UDP 套接字与多重服务器的通信。TimeCompare 是一个 Java 程序，用于从一系列的服务器请求时间，接收其响应，并显示远程系统的时间与本地机的时间之间的差别。

设计该客户程序时最重要的一点是，一个无响应的请求不能挂起程序。用户不能希望每一个请求都得到响应，因此，在调用 receive()之前，要使用 DatagramSocket 实例的 setSoTimeout()方法。

程序清单 32.2 给出了该客户应用程序。

程序清单 32.2 TimeCompare.java

```
import java.io.*;      // Import the package names used.
import java.net.*;
import java.util.*;
import java.text.*;

/**
 * This is an application to obtain the times from
 * various remote systems via UDP and then report
 * a comparison.
 */
public class TimeCompare {
    private static final int TIME_PORT = 13;  // Daytime port.
    private static final int TIMEOUT = 10000; // UDP timeout.
    // This is the size of the datagram data to send
    // for the query - intentially small.
    private static final int SMALL_ARRAY = 1;
    // This is the size of the datagram you expect to receive.
    private static final int TIME_ARRAY = 100;
    // A socket to send and receive datagrams.
    DatagramSocket timeSocket = null;
    // An array of addresses to the machines to query.
    private InetAddress[] remoteMachines;
    // The time on this machine.
    private Date localTime;
    // An array of datagram responses from remote machines.
    private DatagramPacket[] timeResponses;

    /**
     * This method starts the application.
     * @param args Command line arguments - remote hosts.
     */
    public static void main(String[] args) {
      if (args.length < 1) {
          System.out.println(
            "Usage: TimeCompare host1 (host2 ... hostn)");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            System.exit(1);
        }
        // Create an instance.
        TimeCompare runCompare = new TimeCompare(args);
        // Tell it to print out its data.
        runCompare.printTimes();
        System.exit(0);              // Exit.
    }

    /**
     * The constructor looks up the remote hosts and
     * creates a UDP socket.
     * @param hosts The hosts to contact.
     */
    public TimeCompare(String[] hosts) {
        remoteMachines = new InetAddress[hosts.length];
        // Look up all hosts and place in InetAddress[] array.
        for(int hostsFound = 0; hostsFound < hosts.length;
                hostsFound++) {
            try {
                remoteMachines[hostsFound] = InetAddress.getByName(hosts[hostsFound]);
            } catch(UnknownHostException excpt) {
                remoteMachines[hostsFound] = null;
                System.err.println("Unknown host " + hosts[hostsFound] + ": " + excpt);
            }
        }
        try {
            timeSocket = new DatagramSocket();
        } catch(SocketException excpt) {
            System.err.println("Unable to bind UDP socket: " + excpt);
            System.exit(1);
        }
        // Perform the UDP communications.
        getTimes();
    }

    /**
     * This method is the thread of execution where you
     * send out requests for times and then receive the
     * responses.
     */
    public void getTimes() {
        DatagramPacket timeQuery;    // A datagram to send as a query.
        DatagramPacket response;     // A datagram response.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        byte[] emptyBuffer;          // A byte array to build datagrams.
        int datagramsSent = 0;       // # of queries successfully sent.

        // Send out a small UDP datagram to each machine,
        // asking it to respond with its time.
        for(int ips = 0;ips < remoteMachines.length; ips++ ) {
          if (remoteMachines[ips] != null) {
        try {
          emptyBuffer = new byte[SMALL_ARRAY];
          timeQuery = new DatagramPacket(emptyBuffer,
              emptyBuffer.length, remoteMachines[ips],
              TIME_PORT);
          timeSocket.send(timeQuery);
          datagramsSent++;
        } catch(IOException excpt) {
          System.err.println("Unable to send to " + remoteMachines[ips] + ": " + excpt);
        }
      }
    }
    // Get current time to base the comparisons.
    localTime = new Date();
    // Create an array in which to place responses.
    timeResponses = new DatagramPacket[datagramsSent];
    // Set the socket timeout value.
    try {
      timeSocket.setSoTimeout(TIMEOUT);
    } catch(SocketException e) {}
    // Loop through and receive the number of responses
    // you are expecting. You break from this loop prematurely
    // if an InterruptedIOException occurs - that is, if
    // you wait more than TIMEOUT to receive another datagram.
    try {
      for(int got = 0; got < timeResponses.length; got++ ) {
        // Create a new buffer and datagram.
        emptyBuffer = new byte[TIME_ARRAY];
        response = new DatagramPacket(emptyBuffer,emptyBuffer.length);
        // Receive a datagram, timing out if necessary.
        timeSocket.receive(response);
        // Now that you've received a response, add it
        // to the array of received datagrams.
        timeResponses[got] = response;
      }
    } catch(InterruptedIOException excpt) {
      System.err.println("Timeout on receive: " + excpt);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    } catch(IOException excpt) {
      System.err.println("Failed I/O: " + excpt);
    }
    // Close the socket.
    timeSocket.close();
    timeSocket = null;
  }

  /**
   * This prints out a report comparing the times
   * sent from the remote hosts with the local
   * time.
   */
  protected void printTimes() {
      Date remoteTime;
      String timeString;
      long secondsOff;
      InetAddress dgAddr;
      SimpleDateFormat daytimeFormat;

      System.out.print("TIME COMPARISON \n \tCurrent time" +"is: " + localTime +"n \ n");
      // Iterate through each host.
      for(int hosts = 0;
          hosts < remoteMachines.length; hosts++) {
        if (remoteMachines[hosts] != null) {
          boolean found = false;
          int dataIndex;
          // Iterate through each datagram received.
          for(dataIndex = 0; dataIndex < timeResponses.length;
              dataIndex++) {
              // If the datagram element isn't null:
              if (timeResponses[dataIndex] != null) {
                  dgAddr = timeResponses[dataIndex].getAddress();
                  // See if there's a match.
                  if(dgAddr.equals(remoteMachines[hosts])) {
                    found = true;
                    break;
                  }
              }
          }
          System.out.println('Host: ' + remoteMachines[hosts]);
          // If there was a match, print comparison.
          if (found) {
              timeString = new String(timeResponses[dataIndex].getData());
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            int endOfLine = timeString.indexOf("\n");
            if (endOfLine != -1) {
                timeString = timeString.substring(0,endOfLine);
              }
              // Create a SimpleDateFormat object with the time
              // pattern specified.
              //   EEEE - print out complete text for day
              //   MMMM - print out complete text of month
              //   dd   - print out the day in month in two digits
              //   yyyy - print out the year in four digits
              //   HH   - print out the hour in the day, from 0-23 in two digits
              //   mm   - print out the minutes in the hour in two digits
              //   ss   - print out the seconds in the minute in two digits
              daytimeFormat = new SimpleDateFormat("EEEE, MMMM dd, yyyy HH:mm:ss");
              // Parse the string based on the pattern into a
              // Date object.
              remoteTime = daytimeFormat.parse(timeString,new ParseStatus());
              // Find the difference.
              secondsOff = (localTime.getTime() - remoteTime.getTime()) / 1000;
              secondsOff = Math.abs(secondsOff);
              System.out.println("Time: " + timeString);
              System.out.println("Difference: " + secondsOff + " seconds\n");
            } else {
              System.out.println("Time: NO RESPONSE FROM "+ "TOST\n");
            }
          }
        }

      /**
       * This method performs any necessary cleanup.
       */
      protected void finalize() {
          // If the socket is still open, close it.
          if (timeSocket != null) {
              timeSocket.close();
          }
      }
```

32.3.1 启动 TimeCompare

main()方法实例化了一个 TimeCompare 对象，并传递给它与查询主机相对应的命令行参数。main()指示该实例打印出其数据，然后退出。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

TimeCompare 构造器使用 InetAddress.getByName()静态方法查看远程主机，并将返回的 InetAddress 实例放于这些对象的一个数组中。如果查询不到某个主机，该构造器确保元素设置为 null(空)。该构造器在一个动态分配的端口创建一个 DatagramSocket，并最终调用 getTimes()方法执行查询。

32.3.2　getTimes()方法：TimeCompare 的执行路径

该方法所要做的第一件事是，通过 remoteMachines 数组进行迭代处理。对于数组中的每个非空元素，getTimes()方法创建一个小的字节数组，使用该数组构造有适当地址的 DatagramPacket，最后使用 UDP 套接字发送该数据报。当数据报发送后，该方法使用 datagramsSent 跟踪成功发送的查询个数。

现在，一个数据报已经被发送到每一个远程主机，TimeCompare 准备接收响应。此时，getTimes()收集当前时间，用于与远程系统的时间进行比较。接着又创建一个 DatagramPacket 类型的数组，该数组的长度等于成功发送的查询个数，即预期的响应个数。然后 getTimes() 激活 DatagramSocket 实例的 setSoTimeout()方法，以便使服务器响应失败时 accept()不永远阻塞。

getTimes() 接下来进入一个循环，试图对每一个成功发送的查询接收一个 DatagramPacket。当收到响应后，将其放入 timeResponses 数组中，如果 receive()方法设置的时限到达，则中止该循环。

当 getTimes()完成接收响应的循环后，关闭 DatagramSocket。

32.3.3　printTimes()方法：显示比较结果

该方法使用一个 UDP 包数组并打印出其中存放的时间比较结果。在 for 循环内部，将主机与所接收的数据报相匹配。如果找到这样的一个匹配，printTimes()计算出时间的差异，并打印该数据。如果找不到这样的匹配，printTimes()将指出没有收到来自该主机的响应。

32.3.4　运行应用程序

使用 Java 编译器编译 TimeCompare.java，然后使用 Java 解释器执行该程序。TimeCompare 的每个参数应该是包含在比较结果中的远程机器的主机名。例如，要检查自己机器的时间与 www.sgi.com 和 www.paramount.com 的时间，应输入

```
java TimeCompare www.sgi.com www.paramount.com
```

用户所看到的报告应该为：

```
TIME COMPARISON
Current time is: Mon Aug 19 08:03:09 PDT 1996

Host: www.sgi.com/204.94.214.4
Time: Mon Aug 19 08:02:55 1996
Difference: 14 seconds

Host: www.paramount.com/192.216.189.10
Time: Mon Aug 19 08:07:58 1996
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Difference: 288 seconds
```

32.4 使用 IP 多点发送

Internet 协议(IP)是 Internet 上所有信息的传输手段,UDP 数据报被封装在 IP 包中,发送到网上适当的机器。

参见 30.2.1"Internet 协议(IP)"。

大多数 IP 使用单点发送——从一个主机发送一个包到另一个主机。但是,IP 不只限于这种方式,它也具有多点发送的能力。使用多点发送时,一个报文标有一组目标主机地址。一个报文发出后整个组都能收到。

多点发送特别适合于高频带宽的应用程序,例如在网上发送视频和音频数据,因为它不需要建立单独的传输(这样容易使网络饱和)。其它可能的应用包括闲聊对话,分布式数据存储以及在线交互式游戏等。另外,多点发送也可用于客户在网上搜索一个适当的服务器;它可以发送一个多点发送请求,而任何正在侦听的服务器都能与该客户连接并开始一个交互任务。

为支持 IP 多点发送,一定范围的 IP 地址被单独划分出来。这些 IP 地址是 D 类地址,其范围为 224.0.0.0 到 239.255.255.255。这些 IP 地址中的每一个地址都是一个多点发送组。任何标以某组地址的 IP 包都将被已经连接到该组的机器接收到。组的成员是动态成员。

当一个机器与一个多点发送组连结后,它就开始接收发送到该 IP 多点发送地址的报文。现在我们再扩展一下"UDP 套接字特性"一节中的类比,此处的加入一个组类似于建立一个新的邮箱,用来接收打算发送到该组的信息。每一台想加入到一个组的机器都将建立一个自己的邮箱,用来接收相同的信息。如果一个多点发送包发往一个网络,那么正在侦听的任何机器都有机会收到信息。也就是说,使用 IP 多点发送,没有什么机制能限制同一网络上的机器加入一个组。

多点发送组被映射到接口卡硬件地址上。因此,IP 多点发送数据报到达一个不感兴趣的主机时,可迅速被接口卡所遗弃。然而,多个多点发送组映射到一个硬件地址将引起硬件层的不完全过滤,有些过滤仍必须在设备驱动程序层或 IP 层再次执行。

但多点发送也有其局限性——特别是对于通过 Internet 路由器,多点发送数据包的任务。一种特殊的 TCP/IP 协议即 Internet 组管理协议(Internet Group Management Protocol, IGMP),用来管理一个多点发送组的成员。支持多点发送的路由器可使用 IGMP 来决定本地机是否属于某一特殊组;这样的主机采用一个关于使用 IGMP 所加入的组的报告给出响应。根据这些通信,多点发送路由器就可以决定继续转发一个多点发送数据包是否合适。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

警告:要知道,保留一个多点发送组专供自己使用是不存在一种正式的方法的。有些特殊的组被 Internet Assigned Numbers Authority(IAIA) 保留下来作为特殊的用途。这些保留组列在 RFC 1700 中,可通过如下地址获得:

ftp://ftp.internic.net/rfc/rfc1700.txt

除了要避开保留组以外,选择组时还有一些规则。不应使用多点发送路由器在 224.0.0.0 到 224.0.0.255 的组上传送,因为在局域子网上使用它们会限制通信。可试用 224.0.1.27 与 224.0.1.225 之间的任一地址。

如果用户碰巧选择了一个已被使用的组,则他的通信将受到那些机器的干扰。此时用户可退出其应用程序并试着使用其它地址。

除了多点发送组外,多点发送包的另一个重要的方面是 time - to - live(TTL) 参数。TTL 参数用于指定发送者打算使报文传输所经过的独立网络个数。当一个包由路由器向前转发时,包内所包含的 TTL 参数则减去 1,当 TTL 变为 0 时,包就不能再继续被转发了。

提示:选择 TTL 的值时应尽量小。较大的 TTL 值会引起 Internet 网上带宽的不必要浪费。更为重要的是,这很可能会干扰不同区域中碰巧在使用同一组的其它多点发送通信。

如果用户的通信要求限制在该局域网的机器上,应选择 TTL 为 1。当与不在该局域网的机器通信时可以试着估计一下途中要经过的多点发送路由器的个数,然后将 TTL 设置为该数值再加 1。

多点发送主干(Multcast Backbone)或 MBONE 是为了创建一个能够提供多点发送服务的 Internet 路由器网络。但是目前多点发送应用并不普遍。如果所有多点发送的参加者都在同一个物理网络上,就不再需要路由器,从而证明多点发送这一思想是成功的。关于分布式通信更详细的内容,请与网络管理员联系。

32.4.1　Java 的多点发送

Java 的 MulticastSocket 类是使用这一强有力的 Internet 网络特性的关键。MulticastSocket 类允许用户发送和接收使用多点发送 IP 的数据报。要发送一个数据报时,使用缺省的构造器:

```
public MulticastSocket() throws IOException;
```

然后,用户必须创建一个适当格式的 DatagramPacket,地址是介于 224.0.0.0 与 239.255.255.255 之间的多点发送组。当 DatagramPacket 创建之后,可使用 send() 方法进行发送,这需要一个 TTL 值。TTL 值指定允许包所通过的路由器个数。为了防止数据在 Internet 网上大量传播,应避免将 TTL 值设得很大。下面是一个例子:

```
int multiPort = 2222;
int ttl = 1;
InetAddress multiAddr = InetAddress.getByName("239.10.10.10");
byte[] multiBytes = new byte[256];
DatagramPacket multiDatagram = new DatagramPacket(multiBytes, multiBytes.length,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
multiAddr, multiPort);
MulticastSocket multiSocket = new MulticastSocket(); multiSocket.send(multiDatagram, ttl);
```

为了接收数据报,应用程序必须在指定的 UDP 端口上建立一个套接字。然后它还必须加入到接收者的组中。通过使用该套接字,应用程序才能接收 UDP 数据报:

```
MulticastSocket receiveSocket = new MulticastSocket(multiPort);
receiveSocket.joinGroup(multiAddr);
receiveSocket.receive(multiDatagram);
```

当 joinGroup()方法激活后,机器将开始留意网络上所有为该特殊的多点发送组传输的 IP 包,主机还应该使用 IGMP 正确地报告组的使用情况。对于具有多重 IP 地址的机器,用来发送数据报的接口应设置为:

```
receiveSocket.setInterface(oneOfMyLocalAddrs);
```

要想脱离一个多点发送组,可使用 leaveGroup()方法。当通信完成后应当关闭 MulticastSocket:

```
receiveSocket.leaveGroup(multiAddr);
receiveSocket.close();
```

可以看出,使用 MulticastSocket 非常类似于使用一般的 UDP 套接字类 DatagramSocket,二者最根本的差别是:

- DatagramPacket 的地址必须是一个多点发送组。
- MulticastSocket 类的 send()方法使用两个参数:DatagramPacket 和 TTL 值。
- 为了开始侦听多点发送信息,当 MulticastSocket 实例创建之后,必须使用方法 joinGroup()。
- 虽然没有办法像 DatagramPacket 中的 setSoTimeout()一样设置时限,但是在获取流入报文时,receive()方法的使用与 DatagramPacket 中一样。

32.4.2 多点发送应用程序

下面的两个例子给出了一个非常简单的多点发送应用。程序清单 32.3 是一个发送数据报到一个指定多点发送 IP 地址的程序。该程序运行时使用两个参数:第一个指定发送数据报的多点发送 IP 地址,另一个指定侦听应用程序的 UDP 端口。Main()方法保证了这些参数收到后,实例化一个 MultiCastSender 对象。

构造器使用多点发送 IP 地址的 String 对象,创建一个 InetAddress 实例,然后再在动态分配的端口上为发送数据报创建一个 MulticastSocket。然后构造器进入一个 while 循环,从标准输入上逐行读入。该程序将每一行的前 256 个字节包装到一标有适当地址的 DatagramPacket 中,并通过 MulticastSocket 发送该数据报。

程序清单 32.3 MultiCastSender.java

```
import java.net.*;      // Import package names used.
import java.io.*;
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
/**
 * This is a program that sends data from the command
 * line to a particular multicast group.
 */
class MultiCastSender {
   // The number of Internet routers through which this
   // message should be passed. Keep this low. 1 is good
   // for local LAN communications.
   private static final byte TTL = 1;
   // The size of the data sent - basically the maximum
   // length of each line typed in at a time.
   private static final int DATAGRAM_BYTES = 512;
   private int mcastPort;
   private InetAddress mcastIP;
   private BufferedReader input;
   private MulticastSocket mcastSocket;

   /**
    * This starts up the application.
    * @param args Program arguments - <ip> <port>
    */
   public static void main(String[] args) {
      // This must be the same port and IP address used
      // by the receivers.
      if (args.length != 2) {
         System.out.print("Usage: MultiCastSender <IP addr>"
               + " <port> \n\t<IP addr> can be one of 224.x.x.x" + " - 239.x.x.x\n");
      System.exit(1);
      }
      MultiCastSender send = new MultiCastSender(args);
      System.exit(0);
   }

   /**
    * The constructor does all of the work of opening
    * the socket and sending datagrams through it.
    * @param args Program arguments - <ip> <port>
    */
   public MultiCastSender(String[] args) {
      DatagramPacket mcastPacket; // UDP datagram.
      String nextLine;            // Line from STDIN.
      byte[] mcastBuffer;         // Buffer for datagram.
      byte[] lineData;            // The data typed in.
      int sendLength;             // Length of line.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
input =
  new BufferedReader(new InputStreamReader(System.in));
try {
  // Create a multicasting socket.
    mcastIP = InetAddress.getByName(args[0]);
    mcastPort = Integer.parseInt(args[1]);
    mcastSocket = new MulticastSocket();
  } catch(UnknownHostException excpt) {
    System.err.println("Unknown address: " + excpt);
    System.exit(1);
  } catch(IOException excpt) {
    System.err.println("Unable to obtain socket: " + excpt);
    System.exit(1);
  }
  try {
    // Loop and read lines from standard input.
    while ((nextLine = input.readLine()) != null) {
      mcastBuffer = new byte[DATAGRAM_BYTES];
      // If line is longer than your buffer, use the
      // length of the buffer available.
      if (nextLine.length() > mcastBuffer.length) {
        sendLength = mcastBuffer.length;
      // Otherwise, use the line's length.
      } else {
        sendLength = nextLine.length();
      }
      // Convert the line of input to bytes.
      lineData = nextLine.getBytes();
      // Copy the data into the blank byte array
      // which you will use to create the DatagramPacket.
      for (int i = 0; i < sendLength; i++) {
        mcastBuffer[i] = lineData[i];
      }
      mcastPacket = new DatagramPacket(mcastBuffer, mcastBuffer.length, mcastIP, mcastPort);
      // Send the datagram.
      try {
        System.out.println("Sending: \t" + nextLine);
        mcastSocket.send(mcastPacket, TTL);
      } catch(IOException excpt) {
        System.err.println("Unable to send packet: " + excpt);
      }
    }
  } catch(IOException excpt) {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            System.err.println("Failed I/O: " + except);
        }
        mcastSocket.close(); // Close the socket.
    }
}
```

程序清单 32.4 通过接收多点发送的数据报实现了发送者。该应用程序有两个参数，这两个参数必须对应于 IP 地址和用来激活 MultiCastSender 的端口。main() 方法首先检查命令的参数，然后创建一个 MultiCastReceiver 对象。

该对象的构造器用在激活该应用程序的端口上创建一个 InetAddress 和一个 MulticastSocket。在包含于 InetAddress 的地址加入多点发送组，然后进入一个循环。该对象的构造器从一个套接字接收数据报并打印出包含在数据报中数据，包括发送数据包的计算机和端口。

程序清单 32.4　MultiCastReceiver.java

```
import java.net.*;      // Import package names used.
import java.io.*;

/**
 * This is a program that allows you to listen
 * at a particular multicast IP address/port and
 * print out incoming UDP datagrams.
 */
class MultiCastReceiver {
    // The length of the data portion of incoming
    // datagrams.
    private static final int DATAGRAM_BYTES = 512;
    private int mcastPort;
    private InetAddress mcastIP;
    private MulticastSocket mcastSocket;
    // Boolean to tell the client to keep looping for
    // new datagrams.
    private boolean keepReceiving = true;

    /**
     * This starts up the application
     * @param args Program arguments - <ip> <port>
     */
    public static void main(String[] args) {
        // This must be the same port and IP address
        // used by the sender.
        if (args.length != 2) {
            System.out.print("Usage: MultiCastReceiver <IP "
                + "addr> <port> \n\t<IP addr> can be one of "
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            + "224.x.x.x - 239.x.x.x\n");
            System.exit(1);
        }
        MultiCastReceiver send = new MultiCastReceiver(args);
        System.exit(0);
    }

    /**
     * The constructor does the work of opening a socket,
     * joining the multicast group, and printing out
     * incoming data.
     * @param args Program arguments - <ip> <port>
     */
    public MultiCastReceiver(String[] args) {
        DatagramPacket mcastPacket;          // Packet to receive.
        byte[] mcastBuffer;                  // byte[] array buffer
        InetAddress fromIP;                  // Sender address.
        int fromPort;                        // Sender port.
        String mcastMsg;                     // String of message.

        try {
            // First, set up your receiving socket.
            mcastIP = InetAddress.getByName(args[0]);
            mcastPort = Integer.parseInt(args[1]);
            mcastSocket = new MulticastSocket(mcastPort);
            // Join the multicast group.
            mcastSocket.joinGroup(mcastIP);
        } catch(UnknownHostException excpt) {
            System.err.println("Unknown address:" + excpt);
            System.exit(1);
        } catch(IOException excpt) {
            System.err.println("Unable to obtain socket:" + excpt);
            System.exit(1);
        }
        while (keepReceiving) {
            try {
                // Create a new datagram.
                mcastBuffer = new byte[DATAGRAM_BYTES];
                mcastPacket = new DatagramPacket(mcastBuffer, mcastBuffer.length);
                // Receive the datagram.
                mcastSocket.receive(mcastPacket);
                fromIP = mcastPacket.getAddress();
                fromPort = mcastPacket.getPort();
                mcastMsg = new String(mcastPacket.getData());
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            // Print out the data.
            System.out.println("Received from " + fromIP +
            "on port " + fromPort + ": " + mcastMsg);
          } catch(IOException excpt) {
            System.err.println("Failed I/O: " + excpt);
          }
        }
        try {
          mcastSocket.leaveGroup(mcastIP); // Leave the group.
        } catch(IOException excpt) {
          System.err.println("Socket problem leaving group: "+ excpt);
        }
        mcastSocket.close(); // Close the socket.
      }

      /**
       * This method provides a way to stop the program.
       */
      public void stop() {
        if (keepReceiving) {
          keepReceiving = false;
        }
      }
    }
```

要运行该应用程序，首先编译 MultiCastSender 和 MultiCastReceiver。然后，将 MultiCastReceiver 传送到其它机器，以便能够演示多个参与者接收报文的情况。最后，使用 Java 解释器运行该应用程序。

例如，要向 224.0.1.30 组在端口 1111 上发送多点发送报文，应按如下方法进行：

```
-/classes -> java MultiCastSender 224.0.1.30 1111
This is a test multicast message.
Sending:    This is a test multicast message.
Have you received it?
Sending:    Have you received it?
```

要接收这些报文，应在一个或多个系统上运行 MultiCastReceiver 应用程序。这些系统应该加入同一个多点发送组 224.0.1.30，并在相同的端口 1111 上侦听：

```
-/classes -> java MultiCastReceiver 224.0.1.30 1111
Received from 204.160.73.131 on port 32911: This is a test multicastmessage.
Received from 204.160.73.131 on port 32911: Have you received it?
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 33 章　java.net

本章内容：

33.1　URL 类
33.2　URLConnection 类
33.3　HTTPURLConnection 类
33.4　URLEncoder 类
33.5　URLDecoder 类
33.6　URLStreamHandler 类
33.7　ContentHandler 类
33.8　Socket 类
33.9　ServerSocket 类
33.10　InetAddress 类
33.11　DatagramSocket 类
33.12　DatagramPacket 类
33.13　多点发送套接字

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

33.1　URL 类

java.net 软件包提供了低级和高级的网络功能。高级网络类可使用户通过指定信息的类型和位置来访问这些信息。例如,用户可以从 Web 服务器上访问信息。高级网络类所关心的只是那些乏味单调的网络协议,这使得用户可以集中精力于实际的信息。如果用户需要更精确的控制,可使用低级类。低级网络类可使用户在网络上发送原始的数据,实现自己的网络协议。

URL 类的意思是统一资源分配器(Uniform Resource Locator,URL),正如在 Internet 标准 RFC 1630 上定义的那样,它使用的是 World Wide Web 上资源的标准地址格式。一个 URL 类似于一个文件名,它给出了可以获取某些信息的地方,要得到这些信息用户还必须打开并读取这些信息。当用户创建了一个 URL 后,可使用以下三种方法检索存放在该 URL 的信息:

- 使用 URL 类的 getContent 方法直接提取 URL 的内容。
- 使用 openConnection 方法得到一个与 URL 的 URLConnection 连接。
- 使用 openStream 方法得到一个到 URL 的 InputStream 流。

在建立一个 URL 对象时,用户有几种选择。可以使用一个代表完整 URL 的字符串调用构造器:

```
public URL(String fullURL) throws MalformedURLException
```

完整的 URL 字符串可能是用户最为熟悉的形式,例如:

```
URL queHomePage = new URL("http://www.quecorp.com");
```

用户也可以通过给出协议、主机名、文件名及一个可选择的端口号来创建一个 URL:

```
public URL(String protocol, String hostname, String filename)
  throws MalformedURLException
public URL(String protocol, String hostname, int portNumber,
  String filename) throws MalformedURLException
```

使用这种表示方法时,Que 主页的等价格式如下所示:

```
URL queHomePage = new URL("http", "www. quecorp.com","que");
或
URL queHomePage = new URL("http", "www. quecorp.com",80,"que");     //80 是缺省的 http 端口
```

如果用户已经建立了一个 URL,并且想基于已有的 URL 的某些信息打开一个新的 URL,可向构造器传递已有的 URL 及一个字符串:

```
public URL(URL contextURL, String spec)
```

这种方法常用于 Java 小程序中,因为 Applet 类为 Java 小程序的 .class 文件所驻留的目录返回一个 URL。我们也可以得到关于存放 Java 小程序文档的目录的一个 URL。例如,假设我们在存放 Java 小程序的 .html 文件的同一目录下,存放了一个称为 myfile.txt 的文件,则 Java 小程序可按如下方式为 myfile.txt 创建一个 URL:

```
URL myfileURL = new URL(getDocumentBase(), "myfile.txt");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果我们将 myfile.txt 文件存放在 Java 小程序的 .class 文件所在的目录下(可以是也可以不是 .html 文件所在目录),Java 小程序将按如下方式为 myfile.txt 创建一个 URL:

```
URL myfileURL = new URL(getCodeBase(),"myfile.txt");
```

33.1.1 获取 URL 内容

当用户创建了一个 URL 后,可能需要提取其中的内容,最简单的方法是调用 getContent 方法:

```
public final Object getContent()
```

这种方法要求用户为 URL 返回的内容定义一个内容句柄。HotJava 浏览器本身带有一些内置的内容句柄,而 Netscape 并不使用这种方法解释内容。所以,如果在 Netscape 中使用这种方法,很可能会产生一个 UnknownServiceException 异常。

如果我们想自己解释数据,可使用 openConnection 方法为 URLConnection 获取一个 URLConnection 连接:

```
public URLConnection openConnection() throws IOException
```

获得一个 URL 内容的第三种选择方法适用于任何情况。用户可以使用 openStream 方法产生一个 URL 输入流,并自己将其读入:

```
public final InputStream openStream() throws IOException
```

下面的代码段打开一个 URL 输入流并使用一次读一个字节的方式,将一个 URL 的内容复制到 System.out 流:

```
try {
    URL myURL = new URL(getDocumentBase(), "foo.html");
      InputStream in = myURL.openStream();//为 URL 获得输入流
    int b;
    while ((b = in.read()) != -1) {//读下一个字节
        System.out.print((char)b); //打印读取的字节
    }
} catch (Exception e) {
    e.printStackTrace();//出现某种错误
}
```

33.1.2 获取 URL 信息

使用下面的方法,用户可以 URL 的指定部分进行检索:

```
public String getProtocol()
```

该方法返回 URL 的协议名称。

```
public String getHost()
```

该方法返回 URL 的主机名。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

public String getPort()

该方法返回 URL 的端口号。

public String getFile()

该方法返回 URL 的文件名。

public String getRef()

该方法返回 URL 的引用标签。这是 HTML 页面的可选索引项，它在文件名之后，以一个 # 号开始。

33.2　URLConnection 类

与 URL 类中的 getConnection 方法相比，URLConnection 类提供了更详细的 URL 接口。该类所提供的方法有的用于检查 HTTP 头，有的用于获取关于内容的信息，也有的用于获得 URL 输入和输出流。用户可使用的各种不同的协议都有其不同的 URLConnection 类。例如，有一个用来处理 HTTP 协议的 URLConnection 类，也有一个用来处理 FTP 协议的 URLConnection 类。有些浏览器可能不支持其中的任何一个，但是它们都应该能在 HotJava 中实现。HotJava 是完全使用 Java 编写的，它使用这些类完成了所有的浏览操作。Netscape 有其自己的处理这些协议的代码，它并没有使用 Sun 的 URLConnection 类。

URLConnection 类的作用是解释显示于浏览器的文本。因此，它有许多处理头字段和内容类型的方法。

我们不必自己创建 URLConnection 对象。URLConnection 对象是由 URL 对象创建并返回的。当我们有了 URLConnection 实例后，可以使用 getHeaderField 方法检查各种头字段：

public String getHeaderField(String fieldName)

该方法使用 fieldName 返回头字段的值。如果资源中不存在该字段，该方法将返回空(null)。

public String getHeaderField(int n)

该方法返回资源中第 n 个字段的值。如果不存在所指定的头字段，该方法返回空(null)。用户可以使用 getHeaderFieldKey 方法获取相应字段的名字。

public int getHeaderFieldKey(int n)

该方法返回资源中第 n 个字段的字段名。如果头字段的个数小于指定的 n，该方法返回空(null)。

用户也可以以整数或日期的形式获得一个头字段的值，分别使用如下方法：

public int getHeaderFieldInt(String fieldName, int defaultValue)

该方法转换 fieldName 命名的头字段为一个整数，如果该字段不存在或不是一个有效的整数，该方法返回 defaultValue。

public int getHeaderFieldDate(String fieldName long defaultValue)

该方法将头字段值解释成一个日期，并返回从纪元开始到该日期的毫秒数。如果字段不

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

存在或不是一个有效的日期,该方法返回 defaultValue。

除了解释头字段,URLConnection 类还返回关于内容的信息:

```
public String getContentEncoding()
public int getContentLength()
public String getContentType()
```

与 URL 类一样,使用 getContent 方法,用户可以将 URL 作为一对象,获得其完整内容:

```
public Object getContent()throws IOException, UnknownServiceException
```

该方法在 Netscape 下可能不起作用,但是应该可以在 HotJava 中使用。

有时当一个程序试图访问一个 URL 时,需要用户在一个对话框中提供用户证明,这种对话框在用户打开 URL 时会自动弹出。由于我们不总是需要自己的 Java 程序要求用户提供证明,我们可以告知 URLConnection 类,是否允许用户交互。如果出现需要用户交互的情况,而我们已经将其关闭,则 URLConnection 类产生一个异常。

当给 setAllowUserInteraction 方法传送了一个 true(真)值后,如果需要,该方法将允许与一个用户交互:

```
public void setAllowUserInteraction(boolean allowInteraction)

public boolean getAllowUserInteration()
```

该方法返回 ture(真)值,如果需要该类与其它用户交互的话。

```
public static void setDefaultAllowUserInteraction(boolean default)
```

该方法对 URLConnection 的所有新实例,修改允许用户交互的缺省设置。修改缺省设置并不影响已经创建的实例。

```
public static boolean getDefaultAllowUserInteraction()
```

该方法返回允许用户交互的缺省设置。

有些 URL 允许双向通信,通过使用 doInput 和 doOutput 方法,用户可以通知 URLConnection 是否允许进行输入或输出:

```
public void setDoInput(boolean doInput)
public void setDoOutput(boolean doOutput)
```

用户可以将这两个值中的任何一个或全部两个设为 true(真)。doInput 标志的缺省设置为 true(真),而 doOutput 标志的缺省设置为 false(假)。

使用 getDoInput 和 getDoOutput 方法,用户可以查询到 doInput 和 doOutput 标志的设置:

```
public boolean getDoInput()
public boolean getDoOutput()
```

getInputStream 和 getOutputStream 方法返回资源的输入和输出流:

```
public InputStream getInputStream()
  throws IOException, UnknownServiceException
public outputStream getOutputStream()
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
throws IOException, UnknowServiceException
```

33.3　HTTPURLConnection 类

HTTP 协议具有一些 URLConnection 类所没有的额外特性。例如，当用户发送一个 HTTP 请求时，可以进行若干个不同的请求（GET，POST，PUT 等）。HTTPURLConnection 类提供了对 HTTP 专用选项更好的访问方法。

HTTPURLConnection 中最重要的字段之一是 request 方法。用户可以通过使用所需的方法名调用 setRequestMethod，来设置 request 方法：

```
public void setRequestMethod(String method) throws ProtocolException
```

可使用的有效方法包括：GET、POST、HEAD、PUT、DELETE、OPTIONS 及 TRACE。如果用户没有设置请求方法，缺省方法为 GET。调用 getRequestMethod 可返回当前的方法：

```
public String getRequestMethod()
```

当用户发送一个 HTTP 请求时，HTTP 服务器使用一个响应码和报文作为响应。例如，如果用户试图访问一个不存在的 Web 页面，则将获得一个“404 Not found”的信息。getResponseMessage 方法返回响应中的信息部分，而 getResponseCode 方法则返回其中的数字部分：

```
public String getResponseMessage() throws IOException
public int getResponseCode() throws IOException
```

在“404 Not found”中，getResponseCode 返回 404，getResponseMessage 返回“Not found”。

因为 Web 站点经常移动，Web 服务器支持重定向概念，即用户可以自动发送到一个页面的新地址。HTTPURLConnection 类允许用户选择是否需要自动使用重定向。向 setFollowRedirects 方法传送一个 true（真）标志值，将指示 HTTPURLConnection 类使用重定向：

```
public static void setFollowRedirects(boolean flag)
```

如果重定向是打开的，getFollowRedirects 方法返回 true（真）：

```
public static boolean getFollowRedirects()
```

如果所有 HTTP 请求都要经过一个代理，则 getProxy 方法返回 true（真）：

```
public abstract boolean usingProxy()
```

作为 JDK1.2 的新内容，它允许用户获得 HTTP 发生错误后的结果信息流。getErrorStream()方法返回一个 InputStream，它包含了发生错误后所发送的数据。例如，如果服务器响应一个 404，HTTPURLConnection 将发出一个 FileNotFoundException 异常。然而，Web 服务器或许随着 404 发送一个帮助页。getErrorStream()方法为用户提供了该帮助页的句柄。

```
public InputStream getErrorStream()
```

33.4　URLEncoder 类

这个类只包含一个静态方法，该方法用来将一个字符串转换成一个编码的 URL 形式。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

URL 编码时，将一个字符串减少到一个有限的字符集，只有字母、数字及下划线字符不动，空格转换成一个 + 号，所有其它字符转换成一个%xx 形式的十六进制数，此处 xx 为字符的十六进制表示。encode 方法的格式如下：

```
public static String encode(String s)
```

33.5 URLDecoder 类

JDK1.2 增加了一个类，它允许用户对 MIME 格式的字符串进行解码(例如，URLEncoder 所产生的字符串)。当解码过程完成后，ASCII 字符 a 到 z、A 到 Z 以及 0 到 9 保持不变，而加号(+)被转换为空格，字符串中的其它字符则改变成三字符的字符串。这些三字符的字符串以一个百分号开始，后面是两位代表该字符低 8 位的十六进制数字。例如，字符串"Hello + %56"被转换为"Hello V"。

```
public static String decode(String s)
```

33.6 URLStreamHandler 类

URLStreamHandler 类负责分析一个 URL，并创建一个 URLConnection 对象访问该 URL。当用户打开一个 URL 的连接时，它为该 URL 的协议扫描一组包的句柄，这些句柄的名字应为 < protocol > .Handler。例如，如果用户打开一个 HTTP 的 URL，则 URL 类搜索名为 < some package name(某个包的名字) > .http.Handler 的类。缺省情况下，该类只搜索包 sun.net. www. protocol，但也可以通过将系统属性设置为 java.protocol.Handler.pkgs，指定其它的搜索路径。这一系统属性应包括一系列可选择的搜索包，这些包之间用竖线分开。例如：

```
mypackages.urls|thirdparty.lib|funstuff"
```

作为最低限度，URLStreamHandler 的任何一个子类都必须实现一个 openConnection 方法：

```
protected abstract URLConnection openConnection(URL u)
  throws IOException
```

该方法返回一个 URLConnection 实例，它知道如何说明正确的协议。例如，如果用户为 FTP 协议创建了自己的 URLStreamHandler，该方法将返回一个使用 FTP 协议的 URLConnection。

通过建立自己的 parseURL 和 setURL 方法，用户还可以改变分析 URL 字符串的方式：

```
protected void parseURL(URL u,String spec,int start,int limit)
```

该方法分析 URL 字符串时，从字符串中的 start 位置开始，直到 limit 位置结束。当分析完字符串之后，它使用 URL 中的保护性成员 set 方法直接修改 URL。

使用 setURL 方法，可以设置 URL 不同部分的信息：

```
protected void setURL(URL u,String protocol,String host,int port,
  String file,String ref)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

对 set 方法的调用如下所示：

```
u.set(protocol,host,port,file,ref)
```

注意： HotJava 浏览器已经实现了大多数流行的网络协议。如果用户需要在 Netscape 及其它浏览器中使用 URLStreamHandler，则需要自己编写这些实现代码。

33.7　ContentHandler 类

当用户使用 HTTP 提取一个文档时，Web 服务器会在发送实际数据之前，发送给用户一系列的头。这些头中有一项指明了所发送数据的种类。这里的数据称之为内容，而数据的类型（即 MIME 内容类型）由 Content - type 头给出。Web 浏览器使用内容类型决定如何使用流入的数据。

如果用户对某个特殊的 MIME 内容类型提供自己的句柄，可创建一个 ContentHandler 类来分析它，并返回一个代表该内容的对象。设置自己的内容句柄的方法与设置自己的 StreamHandler 的方法基本相同。用户必须给一个形如 < some package name（某个包的名字）> .major.minor 的名字，其中 major 和 minor 名来自于 MIME 的 Content - type 头，其格式如下：

```
Content - type:major/minor
```

major/minor 最常见的组合之一是 text/plain。如果用户定义自己的 text/plain 句柄，可将其命名为 MyPackage.text.plain。缺省情况下，URLConnection 类仅在一个名为 sun.net.www.content 的包中搜索内容句柄。通过在 java.content.handler.pkgs 系统属性中指定一系列由竖线分开的包，用户可以给出其它包的名字。

在自己的 ContentHandler 中，用户唯一必须实现的方法是 getContent 方法：

```
public abstract Object getContent(URLConnection urlConn)
  throws IOException
```

至于实际中如何分析内容和如何选择返回对象的类型，完全取决于用户自己。

33.8　Socket 类

Socket 类是基于网络的 Java 应用程序的基本构造模块之一，它实现了程序之间面向连接的双向通信渠道。当一个套接字连接建立之后，用户可以从该套接字对象获得输入和输出流。要建立一个套接字连接，程序必须时刻侦听一个指定端口号上的连接请求。虽然套接字通信是点对点的——即套接字连接的两端都不看作是主端，数据可以在两个方向上随时发送——但在连接建立阶段仍有服务器和客户这些概念。

设想一个套接字连接为一个电话呼叫。当呼叫完成后，谈话的任何一方都可以随时讲话，但是在最初建立呼叫时，必须有一方呼叫，而另一方则侦听电话铃响。这样，呼叫的一方称为客户，侦听呼叫的一方就是服务器。

本章稍后所讨论的 ServerSocket 类侦听所进入的呼叫，而 Socket 类则初始化一个呼叫。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

网络中等价于号码的是主机地址和端口。主机地址可以是一个主机的名字如 netcom.com，也可以是一个数字地址如 192.100.81.100。端口号是一个 16 位的数字，通常由服务器确定。当用户建立了一个 Socket 对象后，要把将要连接的服务器的目标主机名及端口号传递给构造器。例如：

```
public Socket(String host,int port)
  throws UnknownHostException,IOException
```

该方法用来建立一个主机名由 host 表示，端口号为 port 的套接字连接。如果 Socket 类不能确定主机名的数字地址，将产生一个 UnknownHostException 异常。如果建立连接时有问题——例如，如果没有服务器在指定端口侦听——则将获得一个 IOException 异常。

> **注意**：如果用户想使用数字主机地址建立连接，可以以一个主机名字符串来传递数字地址，例如，主机地址为 192.100.81.100，可以用主机名"192.100.81.100"来传送。

```
public Socket(String host,int port,boolean stream)
  throws UnknownHostException,IOException
```

该方法用来创建一个主机名为 host，端口号为 port 的套接字连接。请求这一连接时，用户可以选择使用基于数据报通信的方式，而不是基于流的方式。使用基于流的方式时，保证了用户通过该连接所发送的所有数据都正确到达。而使用数据报方式则不保证这一点，所以信息有可能丢失。权衡的结果是，数据报方式要比流方式快，如果用户的网络可靠，最好使用数据报方式连接。Socket 对象的缺省方式为 Stream(流)方式。如果为 stream 参数传送 false(假)，连接将以 Datagram(数据报)方式建立。当 Socket 对象建立之后，用户不能改变其模式。

```
public Socket(InetAddress address,int port)
  throws IOException
```

该方法用来创建一个主机地址存储在 address 中，端口号为 port 的套接字连接。

```
public Socket(InetAddress address,int port,boolean stream)
  throws IOException
```

该方法用来建立一个主机地址存储在 address，端口号为 port 的套接字连接。如果 stream 参数为 false(假)，则连接以 Datagram(数据报)方式建立。

> **注意**：由于 Netscape 及其它浏览器的安全限制，可能会限制用户建立到装入 Java 程序的主机地址的套接字连接。

33.8.1 发送和接收 Socket 数据

Socket 类不包含发送和接收数据的显式方法，而只提供了返回输入和输出流的方法，这使得用户能够充分利用 java.io 中现有类的优点。

getInputStream 方法为该套接字返回一个 InputStream 流；而 getOutStream 方法则返回一个 OutputStream 流：

```
public InputStream getInputStream() throws IOException
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public InputStream getOutputStream() throws IOException
```

33.8.2　获取 Socket 信息

用户可以获得关于套接字连接的信息，如地址、与之连接的端口以及其局域端口号等。

> **注意**：正如在一个电话连接中，每个电话都有其自己的电话号码一样，一个套接字的两端也有其主机地址和端口号。然而，客户端的端口号并不输入到连接的建立中。套接字通信与电话之间的一个不同之处是，每次创建一个新连接时客户通常都有一个不同的端口号，但在每次拿起电话拨电话号时，拨号者总是有相同的电话号码。

getInetAddress 和 getPort 方法返回一个连接中另一端的主机地址和端口号：

```
public InetAddress getInetAddress()

public int getPort()
```

从 getLocalPort 方法，用户可以获得套接字连接的局域端口号：

```
public int getLocalPort()
```

33.8.3　设置 Socket 选项

有些套接字选项可改变套接字的行为。这虽然并不常用，但如果打开这些功能将非常有用。setSoLinger 方法用来设置套接字关闭后，试着发送数据将花费的时间量：

```
public void setSoLinger(boolean on,int maxTime)
  throws SocketException
```

一般来说，当用户通过一个套接字发送数据，并关闭该套接字时，所有未传输的数据都将丢失。通过打开 Linger 选项，用户可以保证当所有数据发送完后，再关闭套接字连接。用户可以使用 getSoLinger 查询延迟时间：

```
public int getSoLinger() throws SocketException
```

如果 Linger 选项处于关闭状态，则 getSoLinger 返回 -1。

如果用户试图从一个套接字读取数据，而其中已经没有数据，read 方法一般处于阻塞状态（即它将等待直到有数据）。可使用 setSoTimeout 方法设置 read 方法在放弃读取之前所等待的最大时间量：

```
public synchronized void setSoTimeout(int timeout)
  throws SocketException
```

timeout 为 0 时，read 方法将永远等待下去（缺省行为）。如果 read 超时，而不是正常返回，将发出一个 java.io.InterruptedIOException 异常，但套接字仍保持打开状态。用户可使用 getSoTimeout 查询当前时限：

```
public synchronized int getSoTimeout()
  throws SocketException
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

套接字连接所使用的 TCP 协议在网络利用上效率是相当高的。如果所发送的数据量大，则一般将把数据打成较大的包。这样做之所以效率高，是因为每个网络包都有一个固定的开销量，如果包较大，则由开销所消耗掉的网络带宽的百分比就较小。不幸的是，当用户在很短的时间内发送许多小包时，TCP 会引起延迟。例如，如果用户在网络上发送鼠标坐标，TCP 驱动程序在等待前一个包收到的通知时将频繁地把坐标分组加入较大的包。这将使得鼠标在移动时看起来很不连续。用户可以通过向 setTcpNoDelay 传送 true(真)，要求套接字尽可能快地发送信息：

```
public void setTcpNoDelay(boolean on)
```

如果套接字正在以 No Delay 选项(套接字立即发送信息)运行，那么 getTcpNoDelay 方法将返回 true(真)：

```
public boolean getTcpNoDelay()
```

警告：在使用 No Delay 选项时要特别小心。如果发送很多小的包，将浪费大量的网络带宽；例如，如果发送一个 1 字节的信息，假设固定开销为 64 字节，则所使用带宽中 98% 将用于开销。即使是发送一个 64 字节的信息，还有 50% 的带宽用于开销。

33.8.4 关闭 Socket 连接

等价于“挂掉电话”的是关闭套接字连接，这由 close 方法来执行：

```
public synchronized void close() throws IOException
```

33.8.5 等待到来的数据

从套接字中读取数据与从文件中读取数据有着很大的不同，即使二者都是输入流。在从一个文件中读取数据时，所有的数据都已经在文件中了。而使用套接字连接时，可能在连接的另一端的程序发送信息之前，就已经开始试着读取了。由于 read 方法在不同的输入流中均阻塞——如果没有数据存在就等待，所以用户必须小心，当程序等待时并不是完全挂起。对这种情况，典型的解决办法是产生一个从套接字读取数据的线程。程序清单 33.1 给出了一个专门用于从输入流中读取数据的线程，它通过调用 dataReady 方法，通知程序有新数据到来。

程序清单 33.1 ReadThread.java 的源代码

```
import java.net.*;
import java.lang.*;
import java.io.*;

/**
 * A thread dedicated to reading data from a socket connection.
 */

public class ReadThread extends Thread
{
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    protected Socket connectionSocket; // the socket you are reading from
    protected DataInputStream inStream; // the input stream from the socket
    protected ReadCallback readCallback;

/**
 * Creates an instance of a ReadThread on a Socket and identifies the callback
 * that will receive all data from the socket.
 *
 * @param callback the object to be notified when data is ready
 * @param connSock the socket this ReadThread will read data from
 * @exception IOException if there is an error getting an input stream
 *            for the socket
 */
    public ReadThread(ReadCallback callback, Socket connSock)
    throws IOException
    {
        connectionSocket = connSock;
        readCallback = callback;
        inStream = new DataInputStream(connSock.getInputStream());
    }

/**
 * Closes down the socket connection using the socket's close method
 */
    protected void closeConnection()
    {
        try {
            connectionSocket.close();
        } catch (Exception oops) {
        }
        stop();
    }

/**
 * Continuously reads a string from the socket and then calls dataReady in the
 * read callback. If you want to read something other than a string, change
 * this method and the dataReady callback to handle the appropriate data.
 */

    public void run()
    {
        while (true)
        {
            try {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// readUTF reads in a string
                    String str = inStream.readUTF();
// Notify the callback that you have a string
                    readCallback.dataReady(str);
                }
                catch (Exception oops)
                {
// Tell the callback there was an error
                    readCallback.dataReady(null);
                }
            }
        }
```

程序清单 33.2 给出了 ReadCallback 接口，它必须使用一个接收来自 ReadThread 对象的数据的类来实现。

程序清单 33.2 ReadCallback.java 的源代码

```
/**
 * Implements a callback interface for the ReadConn class
 */
public interface ReadCallback
{
/**
 * Called when there is data ready on a ReadConn connection.
 * @param str the string read by the read thread, If null, the
 *        connection closed or there was an error reading data
 */
    public void dataReady(String str);
}
```

33.8.6 一个简单的 Socket 客户程序

使用上述的这两个类，用户可实现一个简单的客户程序，它连接到一个服务器，并使用一个 read 线程读取该服务器返回的数据。与该客户程序对应的服务器程序在下一节“ServerSocket 类”中提供。程序清单 33.3 给出了 SimpleClient 类。

程序清单 33.3 SimpleClient.java 的源代码

```
import java.io.*;
import java.net.*;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
/**
 * This class sets up a Socket connection to a server, spawns
 * a ReadThread object to read data coming back from the server,
 * and starts a thread that sends a string to the server every
 * 2 seconds.
 */

public class SimpleClient extends Object implements Runnable, ReadCallback
{
    protected Socket serverSock;
    protected DataOutputStream outStream;
    protected Thread clientThread;
    protected ReadThread reader;

    public SimpleClient(String hostName, int portNumber)
    throws IOException
    {
        Socket serverSock = new Socket(hostName, portNumber);

// The DataOutputStream has methods for sending different data types
// in a machine-independent format. It is very useful for sending data
// over a socket connection.
        outStream = new DataOutputStream(serverSock.getOutputStream());

// Create a reader thread
        reader = new ReadThread(this, serverSock);

// Start the reader thread
        reader.start();
    }

// These are generic start and stop methods for a Runnable

    public void start()
    {
        clientThread = new Thread(this);
        clientThread.start();
    }

    public void stop()
    {
        clientThread.stop();
        clientThread = null;
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// sendString sends a string to the server using writeUTF

    public synchronized void sendString(String str)
    throws IOException
    {
        System.out.println("Sending string: " + str);
        outStream.writeUTF(str);
    }

// The run method for this object just sends a string to the server
// and sleeps for 2 seconds before sending another string

    public void run()
    {
        while (true)

            try {
                sendString("Hello There!");
                Thread.sleep(2000);
            } catch (Exception oops) {
// If there was an error, print info and disconnect
                oops.printStackTrace();
                disconnect();
                stop();
            }
        }

// The disconnect method closes down the connection to the server

    public void disconnect()
    {
        try {
            reader.closeConnection();
        } catch (Exception badClose) {
            // should be able to ignore
        }
    }

// dataReady is the callback from the read thread. It is called
// whenever a string is received from the server.

    public synchronized void dataReady(String str)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
            System.out.println("Got incoming string:" + str);
        }

        public static void main(String[ ] args)
        {
            try {
/* Change localhost to the host you are running the server on. If it
   is on the same machine, you can leave it as localhost. */

                SimpleClient client = new SimpleClient("localhost",4331);
                client.start();
            } catch (Exception cantStart) {
                System.out.println("Got error");
                cantStart.printStackTrace();
            }
        }
    }
```

33.9　ServerSocket 类

ServerSocket 类用来侦听所到来的连接，并为每个新连接创建一个 Socket 对象。通过给定一个用来侦听的端口号可创建一个服务器套接字：

```
public ServerSocket(int portNumber) throws IOException
```

如果不关心所使用的端口号，可通过传递一个 0 端口号，使系统自己分配端口号。

许多套接字的实现都有一个连接后备日志的概念。这是指如果许多客户同时连接到一个服务器时，可以被接受的连接个数。当一个服务器达到连接后备日志所限制的连接数时，该服务器将拒绝任何新的客户。要建立一个具有指定的后备日志连接限制的 ServerSocket，可传送端口号及连接后备日志限制数到其构造器：

```
public ServerSocket(int portNumber,int backlogLimit)
  throws IOException
```

注意：由于目前 Netscape 及其它浏览器的安全限制，用户不能使用 Java 小程序接受套接字连接。

33.9.1　接受进入的套接字连接

当服务器套接字建立之后，accept 方法将为每个新的连接返回一个 Socket 对象：

```
public Socket accept() throws IOException
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果所有连接都处理完，则 accept 方法将阻塞直到又有新的连接。如果用户不希望其程序在等待连接时完全阻塞，可在一个独立的线程中执行一个 accept。

当不再需要接受连接时，可使用 close 方法关闭 ServerSocket 对象：

```
public void close() throws IOException
```

close 方法并不影响通过该 ServerSocket 所建立的已经存在的套接字连接。如果要关闭已存在的连接，必须显式地对每个连接进行关闭。

33.9.2 获得服务器套接字地址

若要查找服务器套接字的地址和端口号，可使用 getInetAddress 和 getLocalPort 方法：

```
public InetAddress getInetAddress()
public int getLocalPort()
```

当用户使用系统所分配的端口号时，getLocalPort 方法特别有用。因为我们并不知道系统所分配使用的端口号是多少，但有时候又必须告诉客户所使用的端口号。关于该方法有许多实际的应用，其中之一是实现 FTP 协议。如果用户曾经观察过正在运行的 FTP 会话，可能会注意到，当对一个文件执行 get 或 put 时，会出现一条消息，例如 PORT command accepted。所发生的事情是本地的 FTP 程序创建了一个等价的服务器套接字，并将端口号发送到 FTP 服务器，然后 FTP 服务器使用该端口号创建一个到本地 FTP 程序的连接。

33.9.3 编写一个服务器程序

在编写一个服务器程序时，用户可以使用许多种模型。例如，可以建立一个很大的服务器对象，用来接受新的客户，并且包括了所有与客户通信所必须的方法。也可以通过创建特殊的对象与客户通信，但是这些对象调用主服务器对象的方法，使服务器更加模块化。使用后一种模型，可以使所有客户共享服务器的信息，使用不同的协议进行通信。

程序清单 33.4 给出了一个例子，在该例中客户处理与单个客户通信的对象，并传送客户的请求到主服务器。

程序清单 33.4 ServerCom.java 的源代码

```
import java.io.*;
import java.net.*;

/**
 * This class represents a server's client. It handles all the
 * communications with the client. When the server gets a new
 * connection, it creates one of these objects, passing it the
 * Socket object of the new client. When the client's connection
 * closes, this object goes away quietly. The server doesn's actually
 * have a reference to this object.
 *
 * Just for example's sake, when you write a server using a setup
 * like this, you will probably have methods in the server that
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
 * this object will call. This object keeps a reference to the server
 * and calls a method in the server to process the strings read from
 * the client and returns a string to send back.
 */

public class ServerConn extends Object implements ReadCallback
{
    protected SimpleServer server;
    protected Socket clientSock;
    protected ReadThread reader;
    protected DataOutputStream outStream;

    public ServerConn(SimpleServer server, Socket clientSock)
    throws IOException
    {
        this.server = server;
        this.clientSock = clientSock;
        outStream = new DataOutputStream(clientSock.getOutputStream());
        reader = new ReadThread(this, clientSock);
        reader.start();
    }

/**
 * This method received the string read from the client, calls
 * a method in the server to process the string, and sends back
 * the string returned by the server.
 */
    public synchronized void dataReady(String str)
    {
        if (str == null)
        {
            disconnect();
            return;
        }

        try {
            outStream.writeUTF(server.processString(str));
        } catch (Exception writeError) {
            writeError.printStackTrace();
            disconnect();
            return;
        }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
/**
 * This method closes the connection to the client. If there is an error
 * closing the socket, it stops the read thread, which should eventually
 * cause the socket to get cleaned up.
 **/
        public synchronized void disconnect()
        {
            try {
                reader.closeConnection();
            } catch (Exception cantclose) {
                reader.stop();
            }
        }
    }
```

ServerConn 对象处理了与客户的通信任务，所以服务器对象可以集中精力实现它所应该提供的服务。程序清单 33.5 给出了一个简单的服务器，它收到一字符串，并将该字符串的翻转字符串发送回去。

程序清单 33.5 SimpleServer.java 的源代码

```
import java.io.*;
import java.net.*;

/**
 * This class implements a simple server that accepts incoming
 * socket connections and creates a ServerConn instance to handle
 * each connection. It also provides a processString method that
 * takes a string and returns the reverse of it. This method is
 * invoked by the ServerConn instances when they receive a string
 * from a client.
 */

public class SimpleServer extends Object
{
    protected ServerSocket listenSock;

    public SimpleServer(int listenPort)
    throws IOException
    {
// Listen for connections on port listenPort
        listenSock = new ServerSocket(listenPort);
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void waitForClients()
    {
        while (true)
        {
            try {
// Wait for the next incoming socket connection
                Socket newClient = listenSock.accept();
// Create a ServerConn to handle this new connection
                ServerConn newConn = new ServerConn(
                    this, newClient);
            } catch (Exception badAccept) {
                badAccept.printStackTrace();
                // print an error, but keep going
            }
        }
    }

// This method takes a string and returns the reverse of it

    public synchronized String processString(String inStr)
    {
        StringBuffer newBuffer = new StringBuffer();
        int len = inStr.length();

// Start at the end of the string and move down towards the beginning
        for (int i = len - 1; i >= 0; i--) {

// Add the next character to the end of the string buffer
// Since you started at the end of the string, the first character
// in the buffer will be the last character in the string

            newBuffer.append(inStr.charAt(i));
        }
        return newBuffer.toString();
    }

    public static void main(String[] args)
    {
        try {
// Crank up the server and wait for connection
            SimpleServer server = new SimpleServer(4321);
            server.waitForClients();
        } catch (Exception oops) {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// If there was an error starting the server, say so!
                System.out.println("Got error:");
                oops.printStackTrace();
            }
        }
    }
```

33.10 InetAddress 类

InetAddress 类包含一个 Internet 主机地址。Internet 主机由以下两种方式之一标识：

- 名字
- 地址

地址是一个 4 字节的数字，通常的书写形式为 a.b.c.d，如 192.100.81.100。当数据在计算机之间发送时，网络协议使用该数字地址决定数据应发送的地方。主机名字的创建是为了方便使用。主机名字避免了用户去记忆一个多达 12 位数字的网络地址。例如，记住 netcom.com 要比去记住 192.100.81.100 容易得多。

正如所表明的那样，将一个名字与一个地址关联起来这本身就是一门科学。当用户建立一个与 netcom.com 的连接时，系统需要找出 netcom 的数字地址，此时通常要使用一个称之为域名服务(Domain Name Service，DNS)的服务。DNS 是 Internet 地址的电话簿服务。在 Internet 上主机名和地址被分组成为域和子域，并且每个子域也可能有它自己的 DNS——即它自己的本地电话簿。

我们可能已经注意到，Internet 的主机名通常是由多个句点分隔开的多个名字，这些被分隔开的名字代表着主机所从属的域。例如 netcom5.netcom.com 是 netcom.com 域中一名为 netcom5 的机器的主机名，netcom.com 域是 com 域的子域。netcom.edu 域与 netcom.com 域是完全独立的，netcom5.netcom.edu 将完全是另外一个主域。可以看出，这与电话号码的差别并不大。例如，电话号码 404 - 555 - 1017 有一个区域号码 404，它可以看作是亚特兰大域。交换台 555 是亚特兰大域的一个子域，而 1017 则是 555 域中一个特定的号码，它是亚特兰大域的一部分。正像 netcom5. netcom.edu 与 netcom5.netcom.com 完全不同一样，也可以在另一个区域号码中找到一个相同的电话号码，如 212 - 555 - 1017。

此处应牢记的重要的一点是，一个特定域中主机名是唯一的。读者千万不要以为你的组织是世界上唯一将机器命名为“The Three Stooges”、“Star Trek”及其它有趣的各种字符串的用户。

33.10.1 转换名字为地址

InetAddress 类为用户处理了所有复杂的名字查找操作。getByName 方法使用主机名，返回一个 InetAddress 实例，其中包含有该主机的网络地址：

```
public static synchronized InetAddress getByName(String host)
  throws UnknowHostException
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

一个主机可以有多个网络地址。例如,假设我们有一个自己的局域网以及一个到 Internet 的点对点协议(point - to - point protocol,PPP)连接,那么带有 PPP 连接的那台机器就有两个网络地址:PPP 地址及局域网 LAN 地址。通过调用 getAllByName 方法,可以找出所有的可用网络地址:

```
public static synchronized InetAddress[] getAllByName(String host)
  throws UnknownHostException
```

getLocalHost 方法返回本地网主机的地址:

```
public static InetAddress getLocalHost()
  throws UnknownHostException
```

33.10.2　检验 InetAddress

InetAddress 类有两个用于检索其存储地址的方法。getHostName 方法返回主机的名字,getAddress 方法返回主机的数字地址:

```
public String getHostName()
public byte[] getAddress()
```

getAddress 方法以字节数组返回地址。在目前的 Internet 编址方案下,所返回的数组是一个四字节的数组。但是,如果 Internet 的地址扩充后,该方法将返回一个更大的数组。下述代码段使用圆点分隔形式打印出一个数字地址:

```
byte[] addr = someInetAddress.getAddress();
System.out.println((addr[0]&0xff) + "." + (addr[1]&0xff) + "." +
    (addr[2]&0xff) + "." + (addr[3]0xff));
```

读者可能会感到奇怪,为什么要将地址值与十六进制值 ff 即十进制的 255 进行与运算呢? 原因是 Java 中的字节值是有符号的 8 位数。这意味着当最左边一位为 1 时,该数值为负值。而 Internet 地址通常并不使用负数,通过与 255 进行与运算,并不改变该值,但将其作为一个 32 位的整数值来对待,且其最左边一位为 0,其最右边的 8 位代表 f 地址。

33.10.3　获得一个 Java 小程序的起始地址

在许多可识别 Java 的浏览器下,套接字连接被限制在 Java 小程序所生成的服务器上。换句话说,用户的 Java 小程序能够连接的唯一主机是它被加载的那台机器。用户可以通过获得 Java 小程序的文档或代码基 URL,建立一个与 Java 小程序的生成主机相对应的 InetAddress 实例,然后获得 URL 的主机名。下述的代码段说明了这一方法:

```
URL appletSource = getDocumentBase(); //必须从小程序调用
InetAddress appletAddress = InetAddress.getByName(
    appletSource.getHost());
```

33.11　DatagramSocket 类

Datagram 类实现了一个特殊种类的套接字,是特别为发送数据报而建立的。数据报类似

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

于信件，从一地发送到另一地，并且偶尔会丢失。当然，Internet 的数据报要比邮政系统的信件快若干个数量级。一个数据报套接字很像一个邮箱。用户从数据报套接字中接收所有的数据报。与前面所介绍的基于流的套接字不同，用户不必对每一个必须与之通信的程序都建立一个新的数据报套接字。

如果数据报套接字是邮箱在网络上的对等物，则数据报包就是信件的对等物。当我们需要发送一个数据报到另一个程序时，我们建立一个 DatagramPacket 对象，其中包含着接收 DatagramSocket 的主机地址和端口号，正像寄送信件时必须写上地址一样。然后，调用自己的 DatagramSocket 中的 send 方法，将数据报包发送到以太网上，传送给接收者。

毫不奇怪，在使用数据报时存在一些与邮寄信件时相同的问题，数据报也可能丢失，也可能在投递时顺序错乱。如果我们给某人写了两封信，我们无法保证对方收到信的次序。如果一封信中提及另一封信，这将会引起混乱。对这种情况没有什么好的解决办法，除了对此做出可能的计划。

另外一种情况是数据报丢失。设想一下我们已经寄出了房屋的租金，而一周以后银行又通知我们付款。我们不知道付款出了什么问题——可能是邮件走得很慢，也可能是付款单已经丢失。如果我们再寄一份付款，或许银行最后会收到两张支票。如果我们不再寄另一份，而汇款也确实丢失，那么银行将会很生气。这种情况同样会发生于数据报。我们可能发出了一个数据报而没有任何回应。假设它已经丢失，我们再发送一份，另一端的服务器可能会收到两个请求，从而不知所措。为了最大限度地降低这种情况的发生，最好的方法是在设计应用程序时，使得同样信息的多份数据报不要引起混淆。这种设计的细节不属于本书的讨论范围，用户可参考网络编程方面的书籍。

创建数据报套接字时可以指定端口号，也可以不指定端口号：

```
public DatagramSocket() throws SocketException
public DatagramSocket(int portNumber) throws SocketException
```

与 Socket 类一样，如果不指定端口号，那么系统将自动分配一个端口。只有当其它程序需要发送一份非请求数据报时，才需要使用指定的端口号。无论什么时候发送数据报，其中都有一个返回地址，正如一封信一样。这样，用户发送一份数据报给另一程序时，总能产生一个回应，而无需显式地告诉对方自己所在的端口。一般来讲，只有自己的服务器程序需要一个指定的端口号，发送数据报给服务器和从服务器接收回应信息的客户的端口号都可由系统来分配，因为服务器能够从数据报上看到返回地址。

发送和接收数据报的方法与邮寄信件和检查邮箱一样简单——大部分的工作是书写和阅读信件。send 方法发送数据报到其目的地（目的地存储在 DatagramPacket 对象中）：

```
public void send(DatagramPacket packet) throws IOException
```

receive 方法从一个数据报中读取内容并将所读取的内容存放到一个 DatagramPacket 对象中：

```
public synchronized void receive(DatagramPacket packet)
  throws IOException
```

当用户不再需要数据报套接字时，可使用 close 方法将其关闭：

```
public synchronized void close()
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

最后，如果用户需要知道数据报套接字的端口号，可使用 getLocalPort 方法获得：

```
public int getLocalPort()
```

33.12　DatagramPacket 类

DatagramPacket 类是信件在网络上的对等物。它包含有地址及其它信息。用户创建一个数据报时，必须提供一个包含数据及数据长度的数组。DatagramPacket 类有两种使用方法：

- 作为一个在数据报套接字上发出的数据段：在这种情况下，创建数据包时所使用的数组应该包括所要发送的数据，以及所要发送数据的准确字节长度。
- 作为一个进入的数据报的存放空间：在这种情况下，该数组应该大到足够容纳所希望存放的任何数据，其长度应该是所要接收的字节的最大数。

要创建一个将要发送的数据报，用户不仅必须为数据包给出数据的数组及其长度，而且要提供目的主机及端口号：

```
public DatagramPacket(byte[] buffer,int length,
    InetAddress destAddress,int destPortNumber)
```

当创建一个用于接收数据的数据报包时，用户唯一需要提供的是一个大到足以容纳所进入数据的数组，以及所希望接收的字节的最大数目：

```
public DatagramPacket(byte[] buffer,int length)
```

DatagramPacket 类还提供查询数据包的四个组件的方法：

```
public InetAddress getAddress()
```

对于进入的数据报包，getAddress 返回数据报发送者的地址，对于发出的数据包，getAddress 返回数据报将要发送到的地址。

```
public int getPort()
```

对于进入的数据包，该方法返回的是数据报发送者的端口号，对于发出去的数据包，该端口指的是将要用来发送数据报的端口。

```
public byte[] getData()
public int getLength()
```

33.12.1　广播数据报

广播数据报类似于邮寄宣传品，即同时将一个数据包发送到多个主机。用户在广播时，总是广播到一个指定的端口号，而所广播的网络地址是一个特殊的地址。

回想一下，Internet 的地址是 a.b.c.d 的形式。该地址中的一部分被认为是用户自己主机的地址，而另一部分则是用户所在的网络的地址。网络地址是上述地址中左边的部分，主机地址是其右侧的部分。两者之间的分界线依照地址中第一个字节（上述形式的 a 部分）而变化。如果 a 小于 128，则网络地址就正好是 a 部分，而 b.c.d 是主机的地址。这种地址称之为 A 类地址。如果 a 大于等于 128 并且小 192，则网络地址是 a.b，而主机地址是 c.d，这类地址称之为 B 类地址。如果 a 大于等于 192，则网络地址是 a.b.c，而主机地址是 d，这类地址称之为 C

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

类地址。

为什么网络地址很重要呢？如果要求讲究礼貌的话，那么在广播时应只广播到自己的局域网上。如果广播到全世界将是很不礼貌的，并且也可能并不起作用，因为许多路由器对广播进行封锁，禁止广播到其局域网上。要向局域网上发送一个广播，要使用网络的数字地址，并且将代表主机地址的部分填充上数字 255。例如，如果用户要连接到 Netcom，其网络地址以 192 开始，则只应广播到 Netcom 的 192.100.81 网络上，这意味着数据报的目的地址应该是 192.100.81.255。另外，用户也可能处在 159.165 这样的网上，这是一个 B 类地址，在这个网络上应该广播到 159. 165.255.255。关于这一点，用户可咨询其本地的系统管理员，因为许多 A 类和 B 类网络被本地细化了。如果一定要广播到全体，最安全的方法是广播到 a.b.c.255。

33.12.2 一个简单的数据报服务器程序

程序清单 33.6 给出了一个简单的数据报服务器程序，它只是返回它所收到的数据报。

程序清单 33.6 DatagramServer.java 的源代码

```
import java.net.*;

/**
 * This is a simple datagram echo server that receives datagrams
 * and echoes them back untouched.
 */

public class DatagramServer extends Object
{
    public static void main(String[] args)
    {
        try {
// Create the datagram socket with a specific port number
            DatagramSocket mysock = new DatagramSocket(5432);

// Allow packets up to 1024 bytes long
            byte[] buf = new byte[1024];
// Create the packet for receiving datagrams
            DatagramPacket p = new DatagramPacket(buf,buf.length);
            while (true) {
// Read in the datagram
                mysock.receive(p);

                System.out.println("Received datagram!");

// A nice feature of datagram packets is that there is only one
// address field. The incoming address and outgoing address are
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// really the same address. This means that when you receive
// a datagram, if you want to send it back to the originating
// address, you can just invoke send again.

                    mysock.send(p);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

程序清单 33.7 给出了一个简单的客户程序，它发送数据报到服务器并等待回应。如果数据报丢失，该程序将挂起，因为它并不重发数据报。

程序清单 33.7　DatagramClient.java 的源代码

```
import java.net.*;

/**
 * This program sends a datagram to the server every 2 seconds and waits
 * for a reply. If the datagram gets lost, this program will hang since it
 * has no retry logic.
 */

public class DatagramClient extends Object
{
    public static void main(String[] args)
    {
        try {
// Create the socket for sending DatagramSocket mysock = new DatagramSocket();


// Create the send buffer byte[] buf = new byte[1024];

// Create a packet to send. Currently just tries to send to the local host.
// Change the inet address to make it send somewhere else.

            DatagramPacket p = new DatagramPacket(buf,
                buf.length, InetAddress.getLocalHost(), 5432);
            while (true) {
// Send the datagram
                mysock.send(p);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                System.out.println("Client sent datagram!");
// Wait for a reply
                mysock.receive(p);
                System.out.println("Client received datagram!");
                Thread.sleep(2000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

33.13 多点发送套接字

IP 多点发送(或多点广播)是一种相当新的技术,它是对简单广播的改进。多点广播功能类似于一个将单一信息发送到多个接收者的广播,但仅发送给那些等待它的接收者。

多点广播的思想是,设置一组网络地址作为多点广播地址,其范围是 225.0.0.0 到 239.255.255.255。

> **注意:**实际上,224.0.0.0 与 224.255.255.255 之间的网络地址也是 IP 多点广播地址,但这些地址是作为保留地址供非应用程序使用的。

每一个多点广播地址都被看作是一个组。当用户需要从某个地址接收信息时,加入到该组即可。例如,用户可能为其股票报价系统设置了地址 255.11.22.33 作为多点广播地址,一个想要接收股票查询报价的程序,必须加入到 225.11.22.33 组中。

要发送或接收多点广播数据,必须首先创建一个多点广播套接字。多点广播套接字类似于数据报套接字(事实上,MulticastSocket 是 DatagramSocket 的一个子类)。用户可使用缺省的端口号创建多点广播套接字,也可以在构造器中指定端口号来创建多点广播套接字:

```
public MulticastSocket() throws IOException
public MulticastSocket(int portNumber) throws IOException
```

要加入一个多点广播地址,可使用 joinGroup 方法;要脱离一个组,可使用 leaveGroup 方法:

```
public void joinGroup(InetAddress multicastAddr) throws IOException
public void leaveGroup(InetAddress multicastAddr) throws IOException
```

在某些系统中,可能有多重网络接口。这可能会对多点广播带来问题,因为用户需要在一个指定的接口上侦听。通过调用 setInterface 可选择多点广播套接字所使用的接口:

```
public void setInterface(InetAddress interface)
  throws SocketException
```

例如,如果用户的机器具有 192.0.0.1 和 193.0.1.15 的 IP 地址,并且希望在 193 网络上

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

侦听多点广播信息，可设置接口到 193.0.1.15 地址。当然，用户需要知道该接口的主机名。对于 192 网络，用户可能有主机名为 myhost _ neta，对于 193 网络，主机名为 myhost _ netb，在这种情况下，以如下方式设置接口：

```
mysocket.setInterface(InetAddress.getByName("myhost _ netb"));
```

通过调用 getInterface，可查询多点广播套接字的接口：

```
public InetAddress getInterface() throws SocketException
```

多点发送广播的关键是，用户必须使用一个 TTL(time to live) 值发送出其数据包。TTL 值指定了数据包应该走多远(数据包应该跨过多少个网络)。TTL 为 0 时，指定数据包应停留在本地主机；TTL 为 1 时，指定数据包只能发送到本地网络。另外，TTL 值还有许多特定的含义。TTL 为 32 时，意味着数据包只应发送到本站点的网络上，TTL 为 64 时，意味着数据包应保留在本地区；TTL 为 128 时，意味着数据应保留在本大洲；当 TTL 为 255 时，意味着数据包应发到所有地方。与广播数据报类似，将自己的数据包发送给所有人是不礼貌的。应限制自己数据包的范围为本地网络，或至少为本地站点。

用户在发送多点发送的数据时，使用一个特殊版本的 send 方法，它要用到 TTL 值(若使用缺省的 send 方法，TTL 为 1)：

```
public synchronized void send(DatagramPacket packet,
  byte timeToLive) throws IOException
```

用户还应该牢记，不可靠的 Java 小程序是不允许创建 MulticastSocket 对象的。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 34 章　Java 安全性深析

本章内容：

34.1　Java 安全性的必要性

34.2　Java 安全性框架

34.3　Java 小程序的限制

34.4　Java 安全性问题

34.5　Java 安全性 API：扩展 Java 小程序边界

34.6　安全性 API

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

34.1 Java 安全性的必要性

在所有 Java 环境特性的描述中，都会找到这样的一句话“Java 是安全的”。安全性可以包括许多东西，用户在开发 Java 小程序时，理解 Java 安全性的含义非常关键。用户 Java 小程序的功能被限制在 Java 安全性框架之内，要使得网络加载代码能安全执行，该框架将影响到用户的设计。

为保证一个非常安全的环境，Java 安全性模型采取了严厉的措施。所有加载到网络上的 Java 小程序都被假设具有潜在的敌意，并对之采取了适当的警戒措施。这一事实将极大地限制用户的设计。为了使 Java 小程序超越这些限制，开发出了 Java 安全性 API(Java Security API)。

为了理解 Java 基本框架的目的和基本原理，必须考察使安全性成为一个根本的问题的原因。Java 对安全性提供了多种解决方案，其中很多方案在对于如何使用户在 Internet 网络方案中进行 Java 应用程序的安装及授权，仍存在分歧。

Internet 形成了一个巨大的共享媒介，它允许全世界的机器自由地进行通信。可信任的和不可信任的计算机都连接在一起，允许目的不明的机器访问上百万台计算机。更重要的是，Internet 应用程序和协议并不十分安全，在不同层次上，通过各种技术可以隐藏身份。

将 Java 添加到这样的环境中，将会造成巨大的安全漏洞。Java 的能力引出了许多最有争议的问题，特别是：

- Java 是一个很完善的编程语言，它允许应用程序使用目标计算机上的许多资源，如操作文件，打开到远程系统的网络套接字，以及派生外部进程等。
- Java 代码是从网络上下载的，通常是从一个用户无法控制的计算机上下载的。这样的代码有可能包含致命的错误，或者有可能受到恶意侵入者的修改。程序的编写者有可疑的动机，这也可能对用户系统造成冲突。
- 使用 Java 所允许的浏览器，如 HotJava、Netscape 或 Internet Explorer，可以顺利且完整地下载 Java 代码。这些特殊的程序即 Java 小程序(Applet)可以在用户不知道或没有经其允许的情况下，传送到用户的计算机并执行。

这种“可执行的内容”可能具有超越用户 Web 浏览器设计的最初限制的能力，准确地说，这是因为 Java 的目的就是允许动态地扩展用户浏览器的能力。

给出了这些特征后，很容易看出为什么应该十分小心地对待 Java 代码了。如果没有一个严格控制的环境，大量有疑问的现象就会出现：

- 一段恶意代码毁坏计算机上的文件及其它资源。
- 或许正在运行一个有用的应用程序时，代码悄悄地从系统中检索出敏感数据，并将其传送到攻击者的机器上。
- 当用户仅仅要访问一个 Web 页时，病毒或蠕虫被卸载下来，并从用户的机器继续向其它机器传播。
- 一个程序使用用户的系统作为跳板，攻击其它系统，这样就隐藏起了真正的坏人，并有可能错误地认为该用户就是真正的攻击源。
- 某个程序员所开发的代码由于其能力有限，可能产生程序逻辑陷井，会无意识地毁坏系统。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

记住这些问题，所有这些问题都可能遇到。为了实用，Java 必须提供一个可控制的环境，用来执行应用程序。入侵或无意识毁坏的途径必须事先预测，并加以防止，系统资源必须得到保护。为安全起见，Java 必须假设加载到网络上的代码是来自于一个不可信任的地方；只有那些已经知道是安全的功能才允许进入。但是，Java 不应该如此过分地严格限制，这将使其价值脱离实际。

对那些熟悉 Internet 安全性系统的用户，不应对 Java 所要面临的问题感到陌生。这就提出了一个老的自相矛盾的问题，即计算机必须具有一定功能和可访问的特定资源，然而反过来，提供给系统的功能越强，受攻击的可能性就越大。在这种情况下，采取超越常规的态度将使系统毫无用途，采取容忍的态度会最终招致厄运，而采取谨慎的态度才会找到一条明智的中间道路。

34.2 Java 安全性框架

安全性框架提供了 Java 清晰的 API，以创建安全的运行环境。正如所知，Java 由许多不同的层组成，这些层创建了完整的 Java 运行环境：

- Java 语言
- 富有特性的标准 API
- Java 编译器
- 字节码
- 运行时库的动态装入及检查
- 垃圾收集器
- 字节码解释器

在贯穿于这个结构的关键部分中，特殊的性能帮助保证了执行环境的安全。孤立地来讲每个部分都是没有任何价值或对系统毫无益处的。结合起来看，这些特性会建立一个稳定而安全的框架，它使得 Java 对可执行的内容成为一种实用的解决方案。

34.2.1 第一部分：语言所提供的安全措施

Java 语言本身提供了网络安全的第一层。该安全性为保护数据结构，限制意外程序出错的可能性，提供了必不可少的特性。

Java 严格遵循面向对象的规范 私有数据结构和方法被封装在 Java 类中。只有通过类所提供的接口才能访问这些资源。面向对象的代码通常具有较好的可维护性，并且具有清晰的设计结构。这保证了用户程序不致于突然自己毁坏，或袭击在同一 VM(虚拟机)运行的其它 Java 元素。

无指针的运算 Java 引用不能进行增量运算。或者说，不能重置使其指向 JVM 内存指定位置。这意味着用户不会无意识地覆盖掉专用对象的内容，也不可能将数据写入系统内存的敏感部分。另外，任何并非在等待垃圾收集的对象，必须有一个引用定义，所以对象不会突然丢失。

数组边界检查 Java 中的数组被限制在其所能认识的尺寸之内。所以在 Java 中如果用户有一个数组 int[5]，而试图引用 [8]，则程序将产生一个异常(ArrayIndexOutOfBoundsException)。从历史上看，由于其它语言未能注意到数组的边界问题，出现了大量的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

安全性问题，例如，一个有问题的应用程序可能是由于循环超出了数组边界而引起的，当超越数组边界时，程序将引用到不属于该数组的数据。Java 确保对任何一个企图使用数组索引开始之前或结束之后的元素的行为都发出一个异常，避免了这些问题。

Java 的强制类型转换系统　Java 保证任何一个对象类型到另一个类型的转换操作是合法的。但一个对象不能被任意地转换成另一类型。所以，假设有一个对象如 Thread，若试图将其转换成另一个不兼容的类如 System（即 System s = (System)new Thread()），则运行时将产生异常（ClassCastException）。在语言中不检查类型的兼容性，所以强制两个对象的类型转换，不正确的对象操作，或违反继承限制等都是可能的。

语言对线程安全编程的支持　多线程的编程是 Java 语言的本质之一，特殊的语法保证了不同的执行线程，以顺序和有控制的形式修改关键数据结构。

Final 类和方法：在 Java API 中，有许多类和方法以 final 声明，这防止了程序的进一步派生及特定代码的重载。

34.2.2　第二部分：Java 编译器

Java 编译器将 Java 代码转换成 JVM 字节码，它保证了语言的所有安全特性得以实现。一个值得信赖的编译器建立的代码是安全的，并能保证程序员使用大量的安全特性。编译器确保编译时它所认识的信息是合法的——例如，通过检查强制类型转换错误。

34.2.3　第三部分：检验器

Java 字节码是网上传输的基础，它是 JVM 的机器码。如果仅仅是前面所定义的策略得到满足，Java 安全性仍很容易受到破坏。编写一个怀有恶意的编译器，它将产生可能执行危险动作的字节码，而一个好的 Java 编译器是可以防止这些的。

因此，浏览器端的安全性检查对于保持一个安全的执行环境非常关键。不能设想字节码是从一个友善的编译器如 JDK 中的 Java 产生的，相反，一种不安全的情况是类文件本身就怀有敌意，除非清楚地经过验证。

为了证明这样的断言，当 Java 字节码加载时，首先进入系统的检验器。检验器对加载到 Java 运行环境中的所有类文件执行了一些检查。在批准任何代码加载前，检验器要完成以下几步：

1. 第一个关口是保证类文件具有正常的一般格式。

2. 第二个检查是保证满足各种 Java 约定，例如，检查每个类都有一个超类（Object 类除外），并且 final 类和方法未被重载。

3. 第三步是类文件最详细的检查。在该步中，对字节码自身进行检查，保证其合法性。检验器中的这一机制一般称为字节码检验。

4. 最后一步执行一些附加检查，如保证类字段及方法说明的存在。

> **注意**：有关检验器更详细的信息，请阅读 Frank Yellin 的文章“Low Level Security（底层安全性）”，可从如下地址获得：
>
> http://java.sun.com/sfag/verifier.html。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

34.2.4 第四部分:类装载器

到这一步已确定了字节码是合法的,随后将进入类装载器(Class Loader)。类装载器是抽象类 java.lang.ClassLoader 的一个对象。类装载器加载从网络进入的 Java 小程序,并使其满足 Java 小程序安全性管理器(Applet Security Manager)的限制,Java 小程序安全性管理器将在本章稍后的"第五部分:建立安全性策略"中描述。类装载器准确地为装载到运行系统的类分配类名空间,类名空间是概念上的内存空间,用来存放对象的数据结构。

类装载器保证在未经授权的情况下对象不互相侵入其它对象的类名空间。公共字段和方法可以访问,但是必须定义这样的一个接口,否则其它对象不能看到这些变量。这是很重要的一点,因为系统资源的访问是通过特定的类实现的——这些特定的类是可信赖的,安装于 JDK 之内。如果不可信赖代码能够操作 Java API 内核的数据,将会有灾难性后果。

类装载器也提供对于控制类代码访问的策略网关,例如,Java 小程序禁止重载任何 Java 本身所提供的类,如 Java API 中所提供的那些类。也禁止引入类替代那些允许执行重要系统任务的内部类。当访问对象的引用时,首先检查内部类的类名空间,以防止被网络加载类所欺骗。

34.2.5 第五部分:建立安全性策略

前面所讲的 Java 安全性框架保证 Java 系统不被非法代码和恶意编译器所侵扰。一般地讲,它们保证了 Java 代码按照规则来执行。有了这样的保证,现在就可以建立一个高层的安全性策略了。这种安全性策略存在于应用程序层,允许用户指定 Java 程序可以访问和操作的资源。

Java API 提供了 java.lang.SecurityManager 类,用来创建一个明确定义出一个应用程序能执行和不能执行的任务的集合,如访问文件或网络资源。Java 应用程序并非必须使用 SecurityManager,这意思是所有它所限制的资源都可以自由使用。但是,通过实现 SecurityManager,用户可以增加一个非常重要的保护手段。

支持 Java 的浏览器使用 SecurityManager 来建立安全策略,用以明确划分 Java 小程序和 Java 应用程序所能做的事情。在本章稍后的"SecurityManager 类"一节中将详细讨论这种特殊的限制。

34.2.6 总结

图 34.1 说明了框架中这些单个部分是如何组合在一起,并提供一个安全环境。这个详细的结构为 Java 程序的执行建立了一个智能的自动防止故障装置:

- Java 语言提供使建立安全系统成为可能的特性。
- Java 代码被编译成字节码,此处进行编译类型检查。
- 代码加载到 Java 运行环境,由检验器检查其合法性,这要执行一个多步检查。
- ClassLoader 保证每个加载的类文件有独立的类名空间,允许 Java 解释器实际地执行程序。
- SecurityManager 提供了应用程序层策略,有选择地允许或禁止某些操作。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

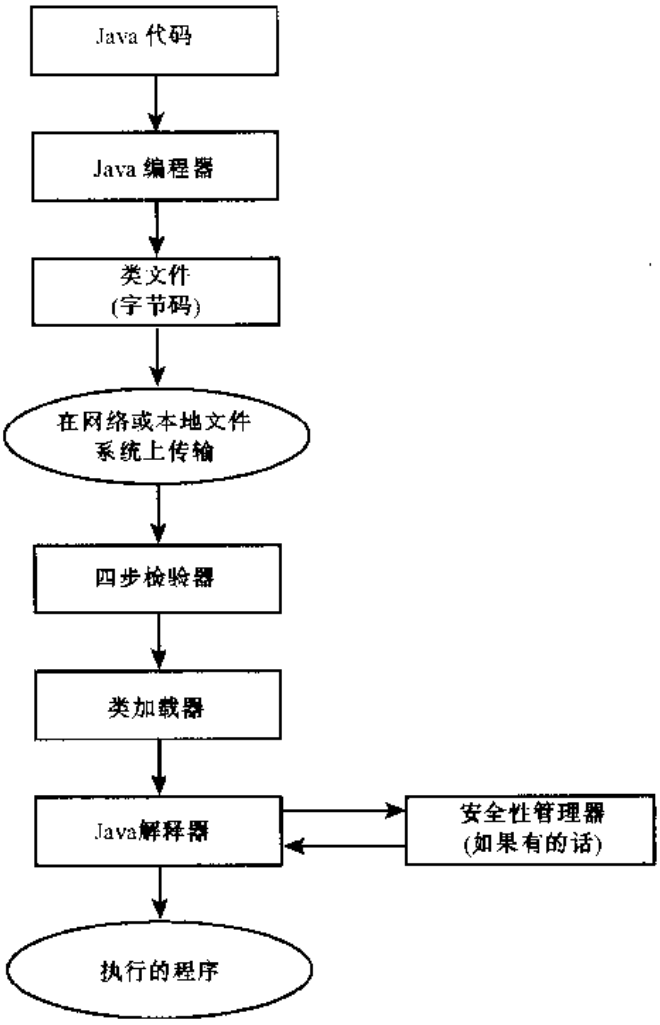


图 34.1　通过不同部分的协调工作创建了一个安全的环境

34.3　Java 小程序的限制

Java 小程序(Applet)是扩展 Java.applet.Applet 类的程序。它可以完全地从支持 Java 的浏览器如 HotJava 或 Netscape 下载并执行。在 JDK1.1 以前,没有建立身份证明和身份信任的机制,所以,所有的 Java 小程序都被假设是来自于一个不可信任的源。

在 JDK1.1 中,Sun 公司增加了可信任 Java 小程序的概念。但是,用户只能以完全可信任或完全不可信任的方式建立 Java 小程序源。JDK1.2 增加了更为精细的控制,允许一个特殊的 Java 小程序源是部分可信任的(所以,可以允许它读取一个特殊文件,而不允许读取其它文件)、完全可信任的或完全不可信任的。

34.3.1　Java 小程序与应用程序

在考察 Java 安全性时应认识的重要的一点是,Java 小程序与 Java 应用程序之间的区别。Java 小程序是扩展 Applet 类的一种特殊程序。只要通过加载含有 APPLET 元素的 HTML 页,便可以在浏览器中动态地执行 Java 小程序。

另一方面,应用程序则是由 Java 解释器直接执行的。它们必须手工地安装于本地系统,由用户在该系统上自觉地执行。Java 浏览器不执行这种程序。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

参见 17.3“开发 Java 应用程序”。

在 JDK1.2 以前，由于 Java 小程序与应用程序之间的区别，允许二者在不同的安全策略下执行。一般认为，在手工安装过程中，用户同意应用程序对系统资源进行访问。对应用程序的信任达到了这样的程度：可以打开并写文件，可以连接到各种网络资源，并且可以执行本地系统的各种程序。这样的一个策略正好与将要安装到个人计算机上的其它应用程序相一致，如果用户愿意，他可以建立一个策略来模仿。但是，事实是所有应用程序都被认为是可信任的，不存在为应用程序建立有限制的安全性策略的方法。JDK 新的安装 API 允许用户为应用程序建立安全性策略，就像对 Java 小程序一样。

34.3.2 SecurityManager 类

增加到 Java 小程序的大多数安全特性是由 java.lang.SecurityManager 类提供的，虽然（像前面提到的一样）ClassLoader 实例的使用也起到了很重要的作用。SecurityManager 类允许用户对特殊的程序建立适合于信任级的特定安全性策略，这一抽象类提供了创建这样一个对象的能力，该对象用来确定是否允许一个程序执行将要执行的操作。

SecurityManager 提供有执行如下操作的方法，用以实现安全策略：

- 确定是否接受一个来自于指定端口上的某个指定主机的网络连接。
- 检查某个线程是否操作另一个 ThreadGrorp（线程组）的线程。
- 检查是否可以在一个指定端口上与一远程系统建立套接字连接。
- 防止建立新的 ClassLoader 类。
- 防止建立新的 SecurityManager，它可能重载现有的策略。
- 检查一个文件是否可以删除。
- 检查一个程序是否可以执行本地系统的一个程序。
- 防止一个程序退出 Java 虚拟机（Java Virtual Machine）。
- 检查是否可以连接一个动态库。
- 检查是否可以在某个网络端口上侦听所到的连接请求。
- 确定一个程序是否可以用指定的 Java 数据包来装载。
- 确定一个程序是否可以在一个指定的 Java 包创建新的类。
- 标记可以使用 System.getProperty()方法访问的系统属性。
- 检查一个文件是否可以读取。
- 检查数据是否可以写入一个文件。
- 检查一个程序是否可以创建自己的网络套接字实现。
- 决定一个程序是否可以创建一个顶层窗口。如果禁止，则任何允许创建的窗口都应包括某种类型的可视警告。

34.3.3 Java 浏览器的安全性策略

在 Web 浏览器内，对于所加载的不信任 Java 小程序已经确定了一种特殊的策略。对于一个程序所允许的所有操作，SecurityManager 都要进行各种检查，装载 Java 类到网上的 ClassLoader 保证从外部系统所加载的类，不对这一安全环境进行破坏。缺省情况下，要使用下述的限制：

- 不允许 Java 小程序在本地系统读取文件。例如，如下的 Java 小程序中的操作将失败：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
File readFile = new File("/etc/passwd");
FileInputStream readIn = new FileInputStream(readFile);
```

● 不允许 Java 小程序在本地系统上创建、修改或删除文件。例如，在 Java 小程序中如下的操作将失败：

```
File writeData = new File("write.txt");          //不能创建文件
FileOutputStream out = new FileOutputStream(writeData);
out.write(1);
File oldName = new File("one.txt");              //不能修改文件，如
File newName = new File("two.txt");              //通过在目录中修改
oldName.renameTo(newName);                       //文件名
File removeFile = new File("import.dat");        //不能删除文件
removeFile.delete();
```

● Java 小程序不能在本地系统上检查一个文件是否存在。例如，在 Java 小程序中如下的操作将失败：

```
File isHere = new File("grades.dbm");
isHere.exits();
```

● Java 小程序不能在本地系统上创建目录。例如，在 Java 小程序中如下的操作将失败：

```
File createDir = new File("mydir");
createDir.mkdir();
```

● Java 小程序不能检查一个目录的内容。例如，在 Java 小程序中如下的操作将失败：

```
String[] fileNames;
File lookAtDir = new File("/users/hisdir");
fileNames = lookAtDir.list();
```

● Java 小程序不能检查各种文件属性，如文件大小、文件类型或上次修改的时间。例如，在 Java 小程序中如下的操作将失败：

```
File checkFile = new File("this.dat");
long checkSize;
boolean checkType;
long checkModTime;
checkSize = checkFile.length();
checkType = checkFile.isFile();
checkModTime = checkFile.lastModified();
```

● Java 小程序不能建立到其它计算机的网络连接，除非是加载该 Java 小程序的计算机这一规则对于使用各种 Java 网络类中的任何一种类所创建的连接都有效，这些 Java 网络类包括 java.net.Socket、java.net.URL 以及 java.net.DatagramSocket。

例如，假设 Java 小程序是从 www.untrusted.org 卸载的，则 Java 小程序中下述的代码将会失败：

```
//不能打开 TCP 套接字
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Socket mailSocket = new Socket("mail.untrusted.org",25);
//URL对象受到类似的限制
URL untrustedWeb = new URL("http://www.untrusted.org/");
URLConnection agent = untrustedWeb.openConnection();
agent.connect();
//就像 UDP 数据报一样
InetAddress thatSite = new InetAddress("www.untrusted.org");
int thatPort = 7;
byte[] data = new byte[100];
DatagramPacket sendPacket =
new DatagramPacket(data,data.length,thatSite,thatPort);
DatagramSocket sendSocket = new DatagramSocket();
sendSocket.send(sendPacket);
```

- Java 小程序不能像网络服务器一样,侦听或接受来自于远程系统的套接字连接。例如,在 Java 小程序中如下的代码将失败:

```
ServerSocket listener = new ServerSocket(8000);
listener.accept();
```

- Java 小程序不能执行任何驻留在本地计算机中的程序。例如,在 Java 小程序中如下的代码将失败:

```
String command = "DEL \ AUTOEXEC.BAT";
Runtime systemCommands = Runtime.getRuntime();
systemCommands.exec(command);
```

- Java 小程序不允许装载动态库或定义内部(native)方法调用。例如,在 Java 程序中如下的代码将要失败:

```
Runtime systemCommands = Runtime.getRuntime();
systemCommands.loadLibrary("lock.dll");
```

- 在 Java 环境中,设置了各种标准系统属性,使用 java.lang.System.getProperty(String key)方法可以访问这些属性。Java 小程序只能读取系统属性中的部分属性,而对其它的属性则不允许访问。表 34.1 给出了这些系统属性。
- Java 小程序不能操作任何 Java 线程,除了那些在其自己的线程组中的线程。
- Java 小程序不能关闭 JVM。例如,Java 小程序中如下的代码将失败:

```
//这种机制失败
Runtime systemCommands = Runtime.getRuntime();
systemCommands.exit(0);
//As does this mechanism.
System.exit(0);
```

- Java 小程序不能创建 SecurityManager 或 ClassLoader 实例。Java 浏览器创建这样的对象,并使用它将安全策略应用于所有 Java 小程序。
- Java.net 数据包使用工厂来建立特定概念的特殊实现,包括协议句柄、内容句柄以及

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

套接字。Java 小程序不能重载这些类的规范：java.net.URLStreamHandlerFactory、java.net.ContentHandlerFactory 以及 java.net.SocketImplFactory。

表 34.1　系统属性与 Java 小程序

关　键　字	目　的	Java 小程序可否访问
file.separator	在文件系统中用于分隔文件和目录的字符(例如，UNIX 中的"/"和 Windows NT/95 中的"\")	是
java.class.path	用于搜索装载的 CLASSPATH 值	否
java.class.version	Java API 所使用的版本	是
java.home	安装 Java 环境的目录	否
java.vendor	用于标识目的厂商指定字符串	是
java.vendor.url	标识厂商资源的 URL	是
java.version	Java 解释器的版本号	是
line.separator	系统中分隔行的字符(例如，UNIX 中的换行符，Windows NT/95 中的换行符和回车符)	是
os.arch	操作系统的硬件体系结构	是
os.name	操作系统的名称	是
os.version	操作系统的版本	是
path.separator	在搜索路径的说明中用于分隔目录的字符(例如，UNIX 中的":"，Windows NT/95 中的";")	是
user.dir	当前工作目录	否
user.home	用户的主目录	否
user.name	用户的帐户名	否

可以想象，这一策略提供了影响用户 Java 小程序能做什么和不能做什么的严格限制。其中一个特殊的问题是，从本质上讲 Internet 是一个分布式的系统，但是 Java 小程序禁止访问这一计算机网——它们只能连接到卸载它们的那台机器上。

另外，由于数据不能写入到本地系统中，Java 小程序在客户端执行时不能保持一个永久性的状态。作为一个工作区，Java 小程序必须连接到存储状态信息的一台服务器上，当以后执行时从原始服务器重新加载信息。

目前的 Java API 对于从已知源加载的可信任 Java 小程序提供了建立特殊安全策略的框架。这一最新的方案将在本章稍后介绍。

34.4　Java 安全性问题

尽管 Java 取得了成功，并引起了广泛的注意，但仍是一个不完全成熟的系统。自从 Java 的第一个版本发布以来，已经发现了很多各种各样的问题，并随之进行了修补。理解这些问题将会使用户有一种似乎身临其境的感觉。

应当注意的最重要的一点是，在 Java 开发领域内一直标榜的它的开放程序。显然，像 sun 及其它一些非常坚决地支持 Java 的公司，当出现程序陷井或问题时应当感受最深。然而，公

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

众的审查批评一直受到鼓励，并且一般地都能很好地接受。

基于大多数安全性专家的经验，这些公众的见解才是安全系统开发的基本构成。在大多数情况下，无法证明一个系统是否是安全的。保险的看法是假设一个没有已知缺陷的系统只是一个等待发现和探寻其缺陷的系统。同样也允许各种专家探索这些隐藏的缺陷——在 Internet 团体内是一个非常熟悉的过程。Java 的进化也遵循了这一思想，从大部分的实际情况观察，这也有益于每个人。

相反的观点认为，向公众公开系统的实现会使不信任和有恶意的个人，在其它人发现问题并纠正这些问题之前，就找出问题，并利用所找出的问题做一些手脚。保持一个系统的机密，将使可恶的黑客发现这些问题的机会大大减少。许多在 Internet 安全性问题上有经验的用户则不同意，他们认为掩盖系统的实现是不明智的：设计保密将创建出一个非常贫乏的系统，同时提供了更多的受到恶意攻击的机会。

警告：给明智者的一句话：对于自以为安全的系统要永远以谨慎的态度对待，这种系统的设计者宣称如果公开其实现的细节，系统的安全将受到侵犯。

34.4.1 已知的缺陷

在 JDK(Java Development Kit)发布后的头几个月里，就发现了一些严重的问题。以下是自从发布以来所发现的部分缺陷的概括介绍：

- 1996 年 2 月，Drew Deam，Edward W. Felton 和 Dan S. Wallach 在 Java Applet Security Manager(Java 小程序安全性管理器)中发现一个问题。这一问题是它不正确地信任了来自于 Domain Name System(域名系统)中的数据，域名系统是 Internet 联系 IP 地址与人类可理解主机名的一种机制。作为一个例子，该缺陷在本节的稍后得到了进一步的验证。

该问题在 Netscape Navigator2.01 和 JDK1.0.1 中通过修补得到解决。

- 1996 年 3 月，Dean，Felton Wallach 发现一个问题，即它允许任意机器码被加载到网络的 Java 小程序执行。这一发现是基于从一个 Java 小程序中加载一个新的类加载器的能力。虽然，JDK 中的 Java 编译器不允许这样的操作，Java 检验器没能防止这一问题。所以，一个有恶意的编译器可以侵扰 Java 安全性框架。当新的类加载器使用 Java 小程序创建之后，能够执行任意机器码。

这一问题在 Netscape Navigator 2.02 和 JDK1.0.2 的修补中得到解决。

- 1996 年 6 月，牛津大学的 David Hopwood，在对象类型强制转换的实现方法中找到一个问题，这一问题是允许任意数据类型之间的强制转换，有了这个缺陷，本地文件可以读取和写入。另外还可以执行任意本地(native)代码。

该问题在 Java 1.1 中纠正。

在所提及的三个缺陷中，被认为是第一的 DNS 攻击受到最普遍的关注。问题的根本在于通过 Security Manager(安全管理器)执行的安全性策略。

由 Web 浏览器所施加的 Java 小程序策略规定，要打开一个新的网络连接。只能是 Java 小程序到其所下载的那台机器的连接。正如在第 30 章“通信和网络”中所指出的，网络计算机在 Internet 上使用 IP 地址相互识别，域名系统(Domain Name System，DNS)允许 IP 地址以各种方式相互联系起来，主要是允许使用人类可理解的主机名。

参见 30.2.1“Internet 协议(IP)”。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在有缺陷的 SecurityManager 中，所进入的 Java 小程序的 IP 地址将用于查看它所映射的 IP 地址集。这种检查应该至少返回最初的 IP 地址，但也可能包含有其它的 IP 地址；这些 IP 地址可以对应于同一个物理计算机，也可以对应于完全独立的计算机。

在设计 Java 小程序时这样的系统可能允许有某些灵活性，允许共享同一个主机名的机器分散处理卸载 Java 小程序初始化的连接。然而，这样的系统以一种非常严重的方式，巧妙的违犯了原始的安全性策略。

DNS 是一种分布式资源，在整个 Internet 上各种系统都有维护特定部分的完整性责任。我们没有能力保证特定的 DNS 服务器不被黑客侵入，怀有恶意的人可以很容易地建立起自己的 DNS 服务器，利用 SecurityManager 这种缺陷提供信息。

从设计上讲，DNS 是不安全的。人们认为 Java 不应为如此一个普遍使用的系统的限制而受责备。Internet 的安全性专家对于 DNS 这种本质是清楚的，所以，对该问题应该有所预见。

关于 Java 安全系统所发现的问题应做出最后的评价，系统的设计从本质上讲是可靠的，它是一个并非完美无缺的设计实现。

34.4.2　拒绝服务攻击

拒绝服务这一术语是描述一种特殊类型安全性攻击的标准方法。这种攻击的目标是不让用户或其它人使用自己的计算机，它并不是企图从用户的系统获取敏感性数据。这种攻击常使用“野蛮的暴力(brute force)”使系统处于超负荷状态。

在区域而并非 Java 中拒绝服务攻击包括如下一些因素：

- 邮件炸弹攻击(mail - bomb attack)，向某人的邮件系统不断地发送大型文档。
- 使用一个如 ping 一样的应用程序使特殊系统泛滥。
- 使用一个自动浏览器重复从 Web 服务器请求资源。

大多数这种攻击利用资源拥护的有效性，使系统瘫痪。因此完全防止系统的这种攻击并非完全有效或可能。只有去掉系统的有用特性，才能保护系统。

拒绝服务攻击使用 Java 小程序也是非常可能的，这些攻击不需要太多的想象：

- 一个 Java 小程序企图大量使用 CPU，使其它应用程序速度降低。
- 一个 Java 小程序不断创建对象，分配越来越多的内存。
- 一个 Java 小程序创建多个窗口，耗尽机器中的 GUI 系统。

目前，这些类型的攻击被划分到 Java 安全性模型的范围之外。Java 必须继续使用下去。如果 Java 小程序具有引人且强有力的能力，它们就可能悄悄地消耗掉算机的实际限制。然而，Sun 公司仍在继续考察更细致地控制 Java 小程序可使用的系统资源的可行性。

34.5　Java 安全性 API：扩展 Java 小程序边界

到现在，我们已经小心地开始认识到对 Java 小程序所施加的缺省限制的重要性。这些策略创造了一个安全但有限制的环境。当用户设计一个 Java 小程序以完成某些任务时，必须建立一个烦杂的工作区，以及一些不能通过 Java 小程序所完成的其它目标。

这种情况是必要的，因为所有的 Java 小程序都被认为是有恶意的——这是一种自动防卫姿态。然而在很多情况下，用户可以断定某些程序并不怀有恶意。例如，一个有信誉的厂商所发行的 Java 小程序，或在自己的防火墙范围内所提供的 Java 小程序。这种程序可以适度地使

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用,它们可以比从某个 Web 页随机下载的 Java 小程序更多地访问系统资源。

在最初的 Java 实现中所忽视的一个关键功能是建立信任关系的能力。Java 1.1 增加了 Java 安全性 API(Java Security API)基类。使用安全性 API,用户具有了创建与程序开发者的信任关系的能力,以及检查来自于这些源代码没有被外部人员更改的能力。然而,Java1.1 的解决方案只是简单地说出一个程序是可信任的或不可信任的;对于一个 Java 小程序究竟可信任到什么程度则没有进一步的控制。在 Java1.2 中,安全模型已经扩展到了可以使用有限信任控制的程度。

Java 安全性 API 的特性是基于计算机的密码设计和算法的,快速地了解一下这些概念将有助于帮助我们理解 Java 安全性 API(Java Security API)的工作原理。

34.5.1 均衡密码技术(Symmetric Cryptography)

对大多数人来说最熟悉的密码技术方案是均衡密码技术,或称为私人键字加密。这一概念是一个特殊的公式或过程,它使用一个数据段和一个特殊密钥,如口令字,来产生一个加密数据块。

只给出加密过的数据或密文,重新生成原始数据的复件是很困难或不可能的。但使用密钥,就可以将密文解码为原始信息。

所以,可以访问密钥的任何人都可以对数据解密。由于系统的安全依赖于这些密钥的保密程度,所以这一方案称之为私人密钥加密。由于用来加密数据的密钥是解密数据所必须的。从本质上讲,这种方案是均衡的。图 34.2 说明了私人保密加密机制。

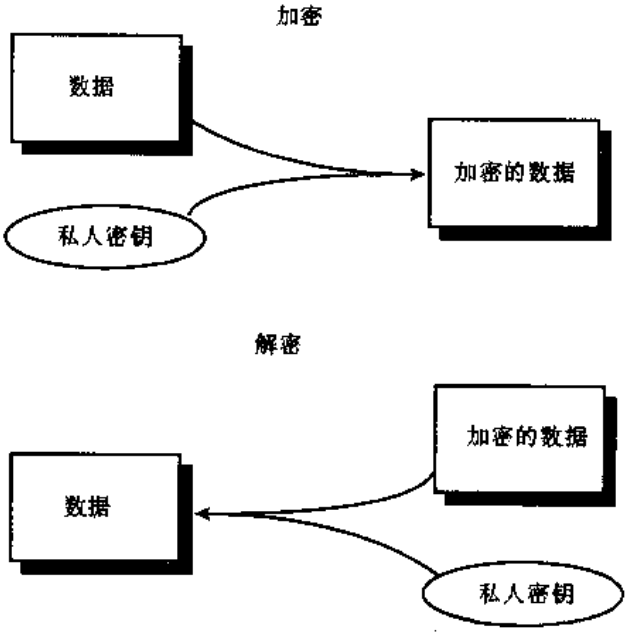


图 34.2 私人密钥加密技术:使用相同的密钥进行加密和解密,为安全起见,密钥必须保密

许多密码系统都使用私人密钥密码技术。数据加密标准(Data Encryption Standard,DES)是一个广泛使用的系统,但是使用今天的技术突破它也是可能的。IDEA 是一种比较新的算法,据认为它比 DES 要安全得多,尽管它尚未像 DES 一样经过全面的测试。RC2 和 RC4 是 RSA 数据安全(Data security)组织推出的两个合适的算法。

使用私人密钥加密技术保持通信的问题之一是,通信双方必须有相同的密钥。然而,这种密钥的交换必须受到保护。所以为了安全地传输文档,必须先有一种交换信息的安全机制。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

34.5.2　公共密钥加密

公共密钥加密是一种表象思想，它是基于 70 代才有所突破的一个基本系统。这一概念基于一种特殊的数学算法。

使用一个特殊的公式创建两个密钥，这两个密钥是数学相关的，但其中的任何一个都不能从另外一个导出。一个密钥用于加密特殊信息，产生密文，另一个密钥用于解密信息，原来的密钥是不能用于解密密文的。因此，这种加密机制是非对称的。

这一系统解决了限制私人密钥加密机制的密钥分散这一问题。一个希望接收被保护文档的用户可以公布其中的一个密钥，一般称之为公共密钥。任何需要向该用户发送加密报文的用户，只要拿到公共密钥就可以创建密文。这种加密报文可以安全地传送，因为只有另外一个密钥可以对它进行解密。接收者使对应的密钥或称秘密密钥对其它人是保密的，因为仅有这一密钥可用于读取由公共密钥加密的报文。图 34.3 给出了这一机制。

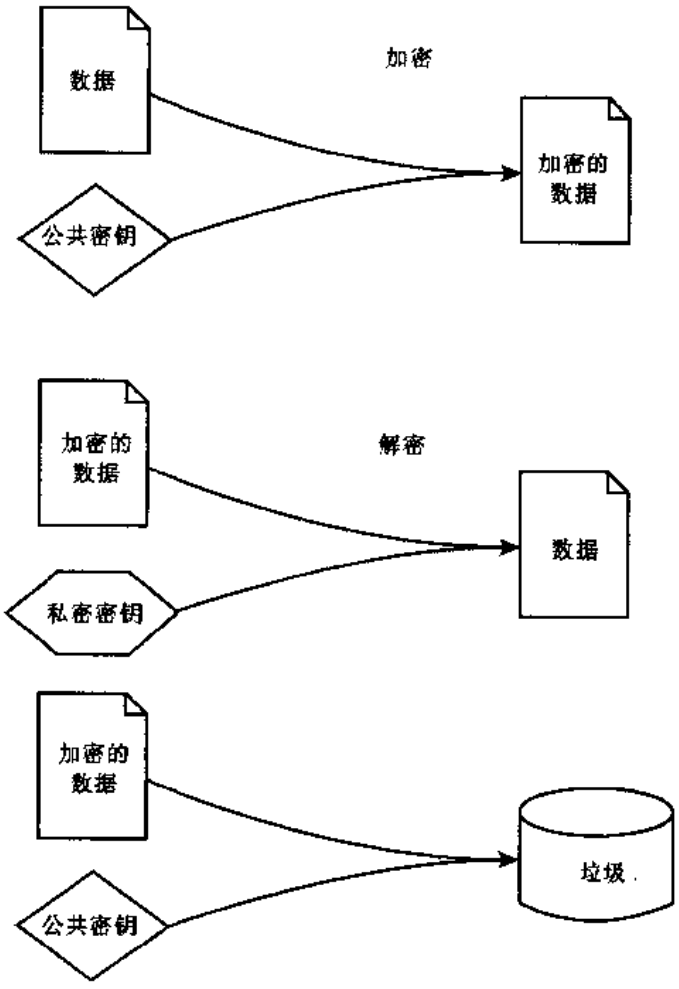


图 34.3　公共密钥加密技术提供了一种密钥分发的解决方案

然而，对 Java 小程序更为有用的也许是反向操作，即签名。给定一个报文，秘密密钥用于创建一个加密的签名。未编码的报文与签名一起传输，并且如果报文被修改，则签名不能被解开。接收报文的任何用户都可以免费获取可以使用的公共密钥，以保证如下两个事件：

- 报文确实来自所期望的作者。
- 报文在签名后未被以任何方式更改过。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用公共密钥加密技术签发报文的过程如图 34.4 所示。

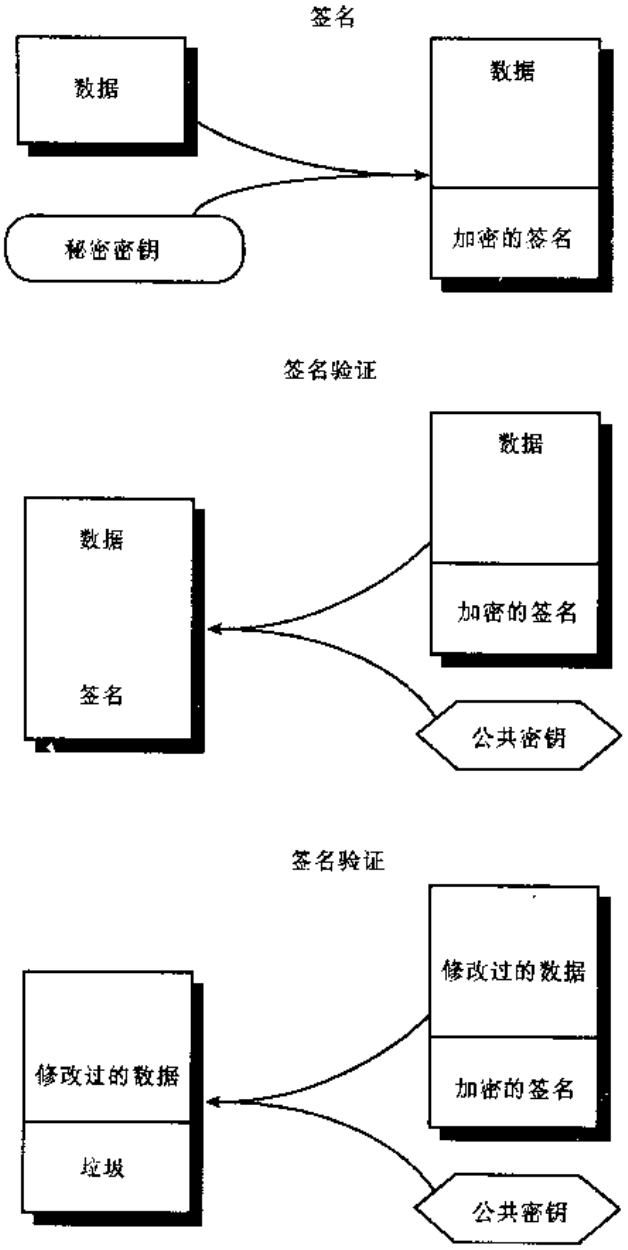


图 34.4 数字签名可建立身份和数据完整性

34.5.3 资格授权

在公共密钥系统中存在的限制之一是要检验公共密钥确实属于我们所确信属于的那个用户。可以想象，一个有恶意的用户也可以发送一个使用秘密密钥签名的报文，然后声称它是来自于我们所确信的用户。随后这种攻击者会以被假冒人的身份公开一个公共密钥，这样，我们会收到该密钥并对签名解密。由于相信已经验证了作者的身份，所以也就相信了那个有恶意的人所写的东西，而这些东西对我们来说是未知的。

Web 的安全传输系统已经开始使用一个称之为 Certification Authorities(CA，身份授权)的系统来克服这种缺陷。基本上可以说 CA 是一个很著名的组织或公司，它为保证公共密钥正常发布做了大量工作。CA 为其它代理签发密钥，以最终证明其身份。当我们接收到这些代理的公共密钥时，可以使用 CA 的公共密钥来进行检验。如果检验成功，则我们将知道 CA 认

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

为该代理是真实的。这样 CA 对该代理进行了验证。

如果我们的 Web 浏览器实现了一种安全通信机制，如 SSL，则我们可以看到一些资格授权列表。Navigator 是支持 SSL 的——如果选择 Options（选项）的 Security Preferences 和 Site Certificates，则可以看到该浏览器中 CAs 所发布的资格列表。

34.5.4　实现了什么

经过以上大量的讨论之后，读者也许并不知道为什么加密可以扩展 Java 小程序的功能。如前所述，Java 小程序被认为是不可信任的和具有潜在敌意的。然而，如果一个 Java 小程序使用公共密钥加密技术进行了数字签名，则可以识别出创建该 Java 小程序的公司，并保证黑客还没有对该公司所编写的内容进行过改动。

现在我们可以建立信任关系了。我们可以为来自于已知代理的 Java 小程序指定特定的角色。例如，我们可能从一个公司购买一个股票报价服务。为使用该服务，我们下载一个 Java 小程序。由于已经与该公司有了一个关系，这就要求我们相信该公司所提供的信息。允许 Java 小程序对本地系统进行更大范围的访问会使人们感到很舒畅：

- 允许 Java 小程序在本地硬盘上保存其配置。
- 允许 Java 小程序连接到位于 Internet 上任何位置的各种股票服务器。
- 允许 Java 小程序将股票信息写入驻留在自己计算机中的一个电子数据表中。

重要的是应当注意到 Java 安全性框架中的其它部分仍然存在。字节码仍然要检验以保证其合法性。另外这并非是一个全有全无的观点。来自于可信任源的 Java 程序可以对自己的计算机有更大的访问权。（回顾一下 SecurityManager 类提供的各种检查，体会允许的访问所增加的程度。）最后，未签名的 Java 小程序仍是不可信任的、它们仍必须遵守 Java Security API 发布之前的那些限制。

34.5.5　密钥管理

密钥管理是安全性问题中非常重要的一个方面，用户必须保持其资格证数据库为最新的，并对其私人密钥保密。如果将密钥资格证随意存放在系统的某个单独的文件中，就有可能偶尔将私人密钥放置在一个公共的目录下，被他人窃取，为了帮助我们管理密钥，Java 1.1 包含了一个密钥数据库和一个密钥管理工具称之为 javakey。现在，在 JDK 1.2 中，数据库和签名代码的功能被分成了两个工具：keytool 和 javasigner。

> **注意**：不幸的是，keytool 和 javasigner 工具与 JDK 1.1 的 javakey 工具不兼容。所以如果由于某种原因仍在使用 1.1，则需要查看 javakey 如何使用，而不是 keytool 和 javasigner。

JDK 1.2 中的 keytool 程序允许用户创建、修改并删除密钥和资格证。它将这些记录存放在一个新型数据库中，该数据库称为 keystore。

资格证是一个数字签名的对象，它由一个已知实体引入，用于识别一个实体的公共密钥例如，用户可能有一个来自 RSA 的资格证，它告诉用户某人的公共密钥。然后当数据被数字签名后，用户可以通过检查资格证验证签名确实是产生于那个人。

在验证签名时用户可以做两件事情。首先，使用资格证检查数据的完整性。数据的完整

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

性指的是自数据被签发后数据没有被修改或损害。其次，验证数据的身份。身份验证是指验证签发报文的人确实是所声称的那个人。

资格证一般由一个实体所持有。一个实体是可以数字签发信息的一个人或一个组织。由于签名需要一个密钥集，签名者需同时有公共密钥和私人密钥，以及一个授权公共密钥的资格证。

密钥是一个数，它与实体联系在一起。公共密钥的设计使每个需要与该实体交互的人可以访问该数字。而私人密钥的设计，只有该实体才知道它。两个密钥是数学匹配的，以便一个使用公共密钥加密的值，可以只使用私人密钥解密。另外，公共密钥和私人密钥的设计，使该私人密钥不能从公共密钥中导出。

为存放一个实体的密钥，用户必须首先在 keystore 数据库中创建一个项。在创建项时必须指定给实体一个名字和一个口令字，用于访问该实体。下述命令为一个名为 mark 的签名者创建一个项：

```
keytool -genkey -alias usingjava -keypass goodbooks
  -dname "cn = QUE" -storepass zippydoda
```

-gendey 选项指明了用户正在为一个签名者创建一个项。-alias 选项指明用户正在为该选项之后直接标识的别名创建一个项。-keypass 用于识别口令字，在任何时候用户访问或修改该密钥时都需要该口令字。

当为一个实体建立完一个项之后，可以为该实体添加密钥和资格证。例如，假设用户在一个名为 vskey.key 的文件中接收到 Verisign 的一个公共密钥，在一个名为 vskey.cer 的文件中接收到该密钥的资格证。在 keytool 命令中使用 -import 标志，将公共密钥导入到密钥数据库：

```
keytool -import -alias verisign -file vskey.cer -keypass verisignps
```

使用 -list 选项，用户可以列出数据库中的实体：

```
keytool -list -storepass zippydoda
  -keytool -list
```

要移去一个实体，使用 -delete 选项：

```
keytool -delete -alias usingjava -storepass abcdefgh
```

34.5.6 为一个 JAR 文件数字签名

当用户自己的 Java 小程序或应用程序保存到一个 JAR 文件时，可以数字签名该 JAR 文件。如果一个 Java 小程序从一个可信任的实体所签名的 JAR 文件加载，则该 Java 程序不遵守通常的安全性限制。在 Java 未来版本中，用户将可以配置所签名的 Java 程序可以做什么，甚至根据谁签发了 Java 小程序而指定许可证。

要签名一个 JAR 文件，首先需要在 keystore 中为一个别名添加一个项，当定义了签名者之后，可使用 jarsigner 工具签发该 JAR 文件。为使用在前一节“密钥管理”中所创建的别名 usingjava 为文件 test.jar 签名，输入：

```
jarsigner -storepass abcdefhg testl.jar usingjava
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

34.5.7　定义策略

为一个 JAR 文件签名，最后的一个疑难问题是建立一个策略文件，定义指定给一个文件许可证。创建一个许可证文件有两种方法。第一种方法是手工建立。程序清单 34.1 给出了如何创建一个策略文件的代码，允许一个由“usingjava”所签字的 JAR 文件写一个称为 newfile 的文件。

程序清单 34.1　write.jp——允许 usingjava 写 Newfile 文件

```
grant SignedBy "usingjava" {
  permission java.util.PropertyPermission "user.home", "read";
  permission java.io.FilePermission "${user.home}/newfile","write";
};
```

编写策略文件的第二种方法是使用 JDK 1.2 所附带的 policytool 工具。policytool 工具的主要用途是简化定义策略文件的工作。

34.5.8　运行 Java 小程序

现在我们已经签名了一个 JAR 文件，定义了一个策略文件，可以使用新的功能运行程序了。要使用 appletviewer 运行 Test 程序，可输入：

```
appletviewer -J-Djava.policy = Write.jp file:///test.html
```

34.6　安全性 API

安全性 API(Security API)的目的是提供对数字签名 Java 小程序的支持。对产生和检查一个程序的数字签名，有几个级别的支持，将来的版本还将提供用于加密和解密信息的类。

安全性 API 存在的原因有两个：允许用户的程序执行安全性功能；允许安全性软件厂商创建自己的安全性提供者服务。在 Java 1.1 中有一个单一安全性提供者，其名字很简单称为“Sun”。其它厂商可能提供他们自己的安全性服务，这些提供者或许会提供 Java 1.2 的 Security API 所定义的服务之外的附加服务。

34.6.1　公共密钥和私人密钥类

安全性 API(Security API)是围绕着密钥而操作的。正如读者所猜测的，安全性 API 为公共密钥和私人密钥均定义了接口。由于这些密钥共享许多的公共特性，它们均从一个称之为 key 的公共超级接口派生出来。密钥有三个重要的特性，包括其算法、格式及编码的密钥值。使用下述方法用户可以检索密钥的这些值：

```
public String getAlgorithm()
public String getFormat()
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public byte[ ] getEncoded( )
```

34.6.2 Signature 类

Signature 类担任着两个不同方面的角色——它可以数字签名一个字节序列，也可以验证一个字节序列的签名。在执行这些功能之前，必须建立一个 Signature 类的实例。Signature 类的构造器是一个保护性成员。创建签名的公共方法称为 getInstance，它使用安全性算法的名字和提供者的名字作为参数：

```
public static Signature getInstance(String algorithm,
  string provider)
```

对 Java1.2 所提供的缺省数据包，getInstance 最常用的调用为：

```
Signature sig = Signature.getInstance("DSA","SUN")
```

如果用户正在创建一个数字签名，使用正在使用的私人密钥调用 Signature 中的 initSign 方法，可创建签名：

```
public final void initSign(PrivateKey key)
```

如果正在检验一个签名，使用待检验的公共密钥调用 initVerify：

```
public final void initVerify(PublicKey key)
```

不管是创建签名还是检验签名，用户必须将所关心的字节序列提供给 Signature 类。例如，如果正在数字签名一个文件，则必须从该文件中读取所有字节，然后将其传送给 Signature 类。update 方法可使用户传送数据字节给 Signature 类：

```
public final void update(byte b)
public final void update(byte[ ] b)
```

update 方法具有可加性，也就是说，每次调用 update 都将增加到现存的将要被签名或被检验的字节数组中。下述代码段从一个文件中读取字节并将其存储在一个 Signature 对象中：

```
Signature sig = new Signature("DSA");
sig.initSign(somePrivateKey);
FileInputStream infile = new FileInputStream("SignMe");
int i;
while ((i = infile.read()) >= 0){
sig.update(i);
}
byte signature[ ] = sig.sign();     //进行签名
```

当字节存放在 Signature 以后，可使用 sign 方法对其进行数字签名，或使用 verify 方法对其进行检验：

```
public final byte[ ] sign( )
public final boolean verify(byte[ ] otherSignature)
```

34.6.3 身份及签名者

正如我们所知，在密钥数据库中存放着两类实体：身份（仅对公共密钥）和签名者（公共/私

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

人密钥对)。Identity 类代表身份,而 Signer 类代表签名者。这两个类都是抽象类,用户不能建立自己的实例。相反,用户必须通过安全性提供者创建并定位这些类。

当用户有了一个 Identety 实例时,可以使用 getPublicKey 检索其公共密钥,或使用 setPublicKey 设置其公共密钥:

```
public PublicKey getPublicKey()
public void setPublicKey(PublicKey newKey)
```

另外,使用 certificates 方法,用户可以检索所有身份的资格证:

```
public Certificate[ ] certificates()
```

使用 addCertificate 和 removeCertificate 方法,用户可以添加或删除资格证:

```
public void addCertificate(Certificate cert)
public void removeCertificate(Certificate cert)
```

Signer 类是 Identity 的子类,它增加了检索私人密钥和设置密钥对的方法:

```
protected PrvateKey getPrivateKey()
protected final void setKeyPair(KeyPair pair)
```

34.6.4　资格证

资格证比数字签名的公共密钥要多一些内容,它包括密钥的拥有者和签名者两部分。拥有者和签名者称为主体(principal),是通常存放于密钥数据库中的实体。使用 getPublicKey 方法,用户可以从一个资格证中检索公共密钥:

```
public abstract PublicKey getPublicKey()
```

用户也可以从一个资格证中检索主体(principal)。Guarantor 是公共密钥签名的实体(保证其可靠性),Principal 是被保证的密钥的拥有者:

```
public abstract Principal getPrincipal()
public abstract Principal getGuarantor()
```

在 Principal 接口中唯一令人感兴趣的方法是 getName,它返回主体的名字:

```
public abstract String getName()
```

34.6.5　IdentityScope 类

IdentityScope 类代表了一个身份集。一般来说,该类代表的是密钥数据库中的身份。当用户有一个 IdentityScope 实例时,可以添加实体、删除实体及查找实体。getSystemScope 方法返回安全系统的缺省身份范围。

```
public static IdentityScope getSystemScope()
```

用户可以通过名字定位身份,也可以通过公共密钥,或使用 Principal 引用定位身份:

```
public Identity getIdentity(String name)
public Identity getIdentity(PublicKey key)
public Identity getIdentity(Principal principal)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

identities 方法返回一个枚举，它允许用户枚举范围之内的所有身份：

```
public abstract Enumeration identities()
```

addIdentity 和 removeIdentity 方法允许用户向本范围内添加新的身份，或删除旧的身份：

```
public abstract void addIdentity(Identity id)
public abstract void removeIdentity(Identity id)
```

程序清单 34.2 给出了为一个文件创建一个数字签名并将该签名写入一个独立文件的示例程序。

程序清单 34.2 SignFile.java 的源代码

```
import java.security.*;
import java.io.*;
import java.util.*;

public class SignFile {
  public static void main(String[] args) {
    try {

      // Get the default identity scope
      IdentityScope scope = IdentityScope.getSystemScope();

      // Locate the entity named trustme
      Identity identity = scope.getIdentity("usingjava");

      // Create a signature and initialize it for creating a signature
      Signature sig = Signature.getInstance("DSA", "SUN");
      Signer signer = (Signer) identity;
      sig.initSign(signer.getPrivateKey());

      // Open the file that will be signed
      FileInputStream infile = new FileInputStream("SignFile.java");

      // Read the bytes from the file and add them to the signature
      int i;
      while ((i = infile.read()) >= 0) {
        sig.update((byte)i);
      }

      infile.close();

      // Open the file that will receive the digital signature of the
      // input file
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            FileOutputStream outfile = new FileOutputStream(
                                                    "SignFile.sig");
            // Generate and write the signature
            outfile.write(sig.sign());
            outfile.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

从一个程序产生数字签名和检验签名的这种功能，允许用户在其程序中提供新的安全级。这在电子商务领域内更为有用，因为这样用户就可以对订单和收据进行数字签名了。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 35 章　对象系列化

本章内容：

35.1　什么是对象系列化
35.2　对象系列化示例
35.3　写和读自己的对象
35.4　定制对象系列化

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

35.1　什么是对象系列化

现在为止我们一直在处理对象，并学会了创建类，以便使用其方法操作对象。然而，当我们必须向一个不同的源，如通过套接字向一个网络或一个文件写一个对象时，只能以内部(native)类型如 int 或 char 的方式写。对象系列化是添加到 Java 的一个工具，它允许我们充分利用 Java 的 OOP 特性，将所建立的对象写到一个文件或其它流中。

为了理解对象系列化，首先来看一个例子，看看如何从另一个源中读入一个简单对象(如字符串)。一般来说，当向一个客户程序或从一个客户程序打开一个流时，差别并不大，只是在发送或接收一个字节。我们可能需要将该字节添加到一个字符串，要完成这一点，其代码或许与程序清单 35.1 中的代码类似。

程序清单 35.1　注意用这种方法产生一个字符串时计算机做了多少工作

```
/*
*
* GetString
*
*/

import java.net.*;
import java.io.*;

public class GetString
{

    //Read in a String from an URL
    public String getStringFromUrl (URL inURL){
        InputStream in;
        try{
            in = inURL.openStream();
        } catch (IOException ioe){
            System.out.println("Unable to open stream to URL:" + ioe);
            return null;
        }
        return getString(in);
    }

    public String getStringFromSocket (Socket inSocket){
        InputStream in;
        try{
            in = inSocket.getInputStream();
        } catch (IOException ioe){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            System.out.println("Unable to open stream to Socket:" + ioe);
            return null;
        }
        return getString(in);
    }

    public String getString (InputStream inStream){
        String readString = new String();
        DataInputStream in = new DataInputStream (inStream);
        char inChar;
        try{
            while (true){
                inChar = (char)in.readByte();
                readString = readString + inChar;
            }
        } catch (EOFException eof){
            System.out.println("The String read was:" + readString);
        } catch (IOException ioe) {
            System.out.println("Error reading from stream:" + ioe);
        }
        return readString;
    }
}
```

在程序清单 35.1 中，最重要的是查看一下 getString()方法。在该方法中我们将看到一个无限 while 循环(产生一个异常时才中止)。如果仔细查看此处究竟发生了什么，就会知道我们正在对字符串的字母，逐个地读并添加，直到到达文件末尾(EOF)。如果没有对象系列化，Java 实际上是没有办法作为一个对象读取一个字符串的。

> **注意**：DataInputStream 确实有 readLine()方法可以返回一个 String，但是这并不是一回事。有两点原因：首先 readLine 并不是从一个整个文件读取；第二，readLine 方法本身实际上与程序清单 35.1 中的 readString()很相似。

当我们需要读取一个由不同类型组成的对象(如程序清单 35.2 中所示)时，更难办的事情出现了。

```
class testObject {
    int x;
    int y;
    float angle;
    String name;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public testObject (int x, int y, float angle, String name){
    this.x = x ;
    this.y = y;
    this.angle= angle;
        this.name = name;
  }
```

如果没有对象系列化要读写 textObject，我们将要打开一个流，读进一块数据，然后使用这些数据去填充一个新对象的内容(通过传送读入的元素到构造器)。我们甚至可以推断出如何读入 textObject 的前三个元素。但是如何读入其中的 name 呢？由于我们刚刚在程序清单 35.1 中编写了一个 readString 类，我们可以使用它，但是我们又如何知道什么时候字符串结束，什么时候下一个对象开始呢？更重要的是，如果 testObject 含有更复杂的引用时又会怎样呢？例如，如果 testObject 如程序清单 35.3 中所示，我们又如何处理从 nextNode 开始的永久性递归呢？

程序清单 35.3　testObject 变得更加复杂

```
class testObject {
    int x;
    int y;
    float angle;
    String name;
    testObject nextNode;
    public testObject (int x, int y, float angle, String name, testObject nextNode){
    this.x = x ;
        this.y = y;
        this.angle= angle;
      this.name = name;
            this.nextNode = nextNode;
    }
}
```

如果我们真的愿意的话，可以编写一个方法(或多个方法)来读和写程序清单 35.3 中的对象，但是如果我们一次能够抓取类的一个完整对象，不是更好吗？

这正是对象系列化所指的内容。在一个不动产程序中，读者使用过一个包含了关于一所房屋的所有信息的类结构吗？没问题——只要打开流并发送或接收整个房屋就行了。读者想保存一个 Java 游戏程序的状态吗？仍然不存在问题，只要将 Java 程序对象发送到流即可。

存储和检索完整对象的能力对于构造大多数临时程序是必不可少的。如果我们需要存储大量的数据，而且常常是过量的数据时，那么使用一个完整的数据库应是我们的需求。即使是需要实现一个数据库，以 BLOB 类型(数据字节流)存储对象也比将其分解为一个 int、一个 char 及一个 byte 来存储要更容易一些。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

35.1.1 对象系列化如何运作

对象系列化的关键是存储关于一个对象的足够多的数据，以便能够完整地重构对象。另外，为了保护用户(和程序员)，对象必须有一个“手印”，以便正确地将对象系列化与用来建立对象系列化的合法对象联系起来。这方面的内容在编写自己的对象时也未讨论过。但对于一个完整的系统来说，它是很关键的。当从一个流中读取一个对象时，其中的每个字段都将被放回到相应位置上的相应类中。

注意：如果读者是一个 C 或 C++ 程序员，可能习惯于对一个类或结构使用指针，并使用 sizeOf()方法然后写出整个类，通过这样的方式完成大部分的对象系列化工作。不幸的是，Java 不支持指针或直接内存访问，所以这种技术在 Java 中不起作用。但对象系列化仍是必不可少的。

然而，对一个系列化系统来说，存储一个类的方法或临时字段是不必要的。类代码在任何需要它的时候都是可访问的。换句话说，当用户恢复 Date 类时，并不恢复方法 getHours()。它假设用户已经把 Date 的值恢复到 Date 对象中，并且该对象有 getHours()方法所需的代码。

35.1.2 使用对象引用处理对象

一个对象经常以类变量(字段)的形式引用其它对象。换句话说，在更复杂的 testObject 类(参见程序清单 35.3)中添加了 nextNode 字段，该字段是该对象中的一个对象引用。为了保存一个类，也必须保存这些引用对象的内容。当然，引用对象也可能还引用了更多的对象(例如如果 nextNode 还有合法的 nextNode 值的话、对于 testObject)。所以作为一条规则，要完全系列化一个对象，必须存储该对象的所有信息，以及每一个该对象可达的对象，包括所有递归对象。

警告：对象系列化和远程方法调用(Remote Method Invocation)在 Netscape Navigator 3.0、Microsoft Internet Explorer 3.0 或 Java Development Kit 1.0 下，是不能使用的。第一个支持对象系列化和 RMI 的 JDK 版本是 JDK 1.02，但这只是一个修补版本。在试用这些特性时，推荐使用 JDK 1.1。

如果用户正在使用 JDK 1.02，那么必须分别获取对象系列化和 RMI 类，并且还必须将其添加到现有类库中。这些类可从下述 URL 地址下载：

http://chatsubo.javasoft.com/current/download.html

35.2 对象系列化示例

作为一个简单示例，我们向一个文件存储或从一个文件中检索 Date 类。如果没有对象系列化功能，要完成这些操作，用户可能要在 getTime() 的顺序上做一些工作，并将长整型结果写到文件中。然而，使用对象系列化，这一过程将非常简单。

35.2.1 一个写 Date 类的应用程序

程序清单 35.4 给出了称为 DateWrite 的示例程序。DateWrite 创建一个 Date 对象，并将

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

整个对象写入一个文件。

程序清单 35.4　DateWrite.java——写 Date 对象到一个文件的应用程序

```
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.util.Date;
public class DateWrite {
        public static void main (String args[]){
            try{
                // Serialize today's date to a file.
                FileOutputStream outputFile = new
                FileOutputStream("dateFile");
                ObjectOutputStream serializeStream = newbjectOutputStream(outputFile);
                serializeStream.writeObject("Hi!");
                serializeStream.writeObject(new Date());
                serializeStream.flush();
            } catch (Exception e) {
                System.out.println("Error during serialization");
            }
        }
}//end class DateWrite
```

让我们看一下程序清单 35.4 中的代码，首先，注意该程序创建了一个 FileOutputStream。为了进行系列化，必须首先声明一个将要关联到 ObjectOutputStream 的某种类型的 OutputStream。（正如在程序清单 35.5 中所看到的，也可以使用从任何其它对象，包括一个 URL 所产生的 OutputStream。）

注意：要使用 JDK 1.02 编译程序清单 35.4，需要增加一些额外的命令。我们对这些命令可能并不习惯，在开始之前，首先检查一下是否已经下载了 RMI 和对象系列化类，并将文件解压缩到 Java 目录下。现在，输入下述命令：

```
java -classpathc:\java\lib\classes.zip;c:\java\lib\
  objio.zip;.DateWrite.java
```

在上述编译命令中，假设用户所使用的是一个 Windows 系统，并且存放 Java 文件的目录是 C:\JAVA。如果文件存放在其它位置或所使用的系统不是 Windows，则需要使用 Java 安装的正确路径来替换 C:\JAVA\LIB。查看安装所包含的 README 文件，并阅读发行注释中有关已知陷井或问题的内容，总是一个好注意。

上面在编译 DateWrite 时应该是没有错误的。如果出现错误，确认在 JAVA\LIB 目录下有一个 OBJIO.ZIP 文件。同时，要确认已经在类路径中包括了 CLASSES.ZIP 和 OBJIO.ZIP 文件。

当创建一个流后，使用该流创建一个 ObjectOutputStream。ObjectOutputStream 包括了系

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

列化任何对象和写对象到流所必须的所有信息。

在上述不长的代码段中，我们看到两个对象正在被写入流中，所写的第一个对象是 String 对象，第二个则是 Date 对象。

35.2.2 在 JDK 1.02 下运行 DateWrite

一旦编译完 DateWrite 程序，则可以运行它了。然而，正如在编译 DateWrite 类时一样，必须在路径中包含 OBJIO.ZIP 文件。同时，要确认已经在类路径中包括了 CLASSES.ZIP 和 OBJIO.ZIP 文件。

```
Java -classpath c:\java\lib\classes.zip;c:\java\lib\
  objio.zip;.DateWrite
```

注意：如果在类路径中未包含 OBJIO.ZIP 文件，则可能得到一个错误，如：

```
java.lang.NoClassDefFoundError:java/io/ObjectOutputStream
  at DateWrite.main(DateWrite.java:9)
```

这是虚拟机不能定位对象系列化所需类文件的结果。

35.2.3 编译和运行 DateWrite

为使用 JDK 1.1 编译和运行 DatetWrite，复制程序清单 35.4 的内容到一个名为 DateWrite 的文件，并像对其它文件一样，使用 javac 编译该文件：

```
javac DatetWrite.java
```

同样，就像运行其它 java 应用程序一样，运行该程序：

```
java DatetWrite
```

DatetWrite 类并不产生实际输出，所以当运行该程序时应该是很快地返回到命令提示符。但是，如果现在查看一下目录结构，应该看到一个文件 dataFile。这是刚刚建立的文件。如果试图显示出该文件内容，将会看到一些令人费解的信息。

然而，仔细考察就会发现该文件包含了一些内容，这些看上去令人费解的东西，实际上是系列化所使用的关于该类的存储信息，如字段的值以及前面所讨论的类签名。

35.2.4 一个读入 Date 类的简单应用程序

当然，下一步就是从文件中读取 Date 和 String。让我们来看看它能复杂到什么程度。程序清单 35.5 给出了读入 String 和 Date 的示例程序。

程序清单 35.5 DateRead.java——读回 String 和 Date 的应用程序

```
import java.io.FileInputStream;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.io.ObjectInputStream;
import java.util.Date;
public class DateRead {
        public static void main (String args[]){
                Date wasThen;
                String theString;
                try{
                    // Serialize date from a file.
                    FileInputStream inputFile = new FileInputStream("dateFile");
                    ObjectInputStream serializeStream = new
                ObjectInputStream(inputFile);
                    theString = (String) serializeStream.readObject();
                } catch (Exception e) {
                    System.out.println("Error during serialization");
                    return;
                }
                System.out.println("The string is:" + theString);
                System.out.println("he old date was:" + wasThen);
        }
}
```

程序清单 35.4 和 35.5 的主要区别是明鲜的。程序清单 35.4 是写，而程序清单 35.5 是读。在 DateRead 中，我们首先声明了两个变量用来存储对象。要记住这一点，因为如果在 try - catch 块中创建这些变量，则在到达 System.out 行之前，它们就可能超出作用范围。其次，创建了一个 FileInputStream 和 ObjectInputStream，就像在 DateWrite 中所创建的 FileOutputStream 和 ObjectOutputStream 一样。

代码中的下两行也较为清楚，但应特别注意强制类型转换操作符。readObject()返回一个 Object 类。缺省情况下，Java 不多态强制转换任何对象，所以在进行转换时必须显式指定。余下的代码就好理解了。

我们可以按照对 DatetWrite 的介绍，编译并运行 DateRead，这应是很简单的。

注意：要使用 JDK1.02 编译该代码，可设置一个 classpath 变量，以避免每次使用 javac 时都使用 - classpath 选项。也可以像在前面的例子中那样，使用 - classpath 选项，但是设置 classpath 变量的办法更有效一些。这两种方法是可以互换的。设置 classpath 变量的方法如下：

在 windows 系统下，输入：

```
set classpath = c:\java\lib\classes.zip;c:\java\lib\objio.zip;.
```

在其它平台下，其语法格式略有差别。例如，在 UNIX 下可使用如下格式：

```
classpath = /usr/java/lib/classes.zip:/usr/java/lib/objio.zip;.
    export classpath
```

不管在那种系统下，不要忘记添加当前目录(.)到 classpath 语句的末尾处。在没有列出当前目录的情况下，javac 仍可以运行，但 java 命令将无法执行。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

35.2.5 编译和运行 DateRead

现在可以编译并运行 DateRead 了。只要按照 DateWrite 中所介绍的方法进行即可。

```
javac DateRead.java
```

使用熟悉的 java 命令也可以运行该程序：

```
java DateRead
```

下面是这些代码输出结果的一个例子：

```
The String is: Hi!
The old date was: Wed Dec 1 23:36:26 edt 1996
```

注意 String 和 Date 正是以它们写出时的形式被读进来。现在我们可以使用流写出和读入整个对象，而不需再分别将每个元素压入流中了。

警告：正象读者所预料的，读入对象时的顺序必须与写出时一致。如果不一致，将发生一个运行时错误，其提示信息如下所示：

```
Error during serialization
```

35.2.6 使用 Java 小程序读入 Date 类

对象系列化并非仅限于应用程序。程序清单 35.6 给出了 DateRead 的变型，以便于它也能作为一个 Java 小程序运行。

程序清单 35.6 DateReadApp.java——从文件中读取 Date 对象的 Java 小程序

```
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.util.Date;
import java.awt.Graphics;
public class DateReadApp extends java.applet.Applet {
    public void paint (Graphics g){
        Date wasThen;
        String theString;
        try{
            // Serialize date from a file.
            FileInputStream inputFile = new FileInputStream("dateFile");
            ObjectInputStream serializeStream = new
        ObjectInputStream(inputFile);
            theString = (String) serializeStream.readObject();
            wasThen = (Date)serializeStream.readObject();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        } catch (Exception e) {
            System.out.println("Error during serialization");
            return;
        }
        g.drawString(("The string is:" + theString),5,100);
        g.drawString(("The old date was:" + wasThen),5,150);
    }
}
```

当编译完程序清单 35.6 后，输出结果应如图 35.1 所示。切记必须使用 Java 的 Applet Viewer 运行该程序，因为其它的浏览器尚未支持对象系列化。

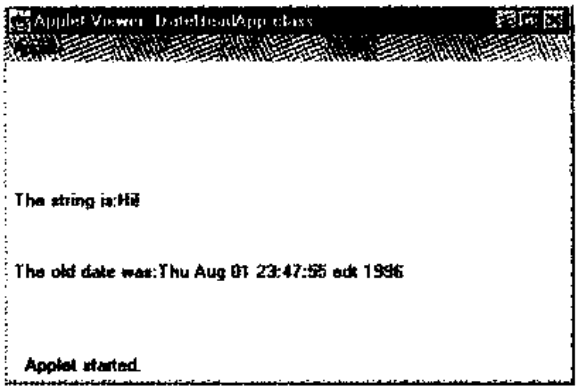


图 35.1　使用系列化读入 Date 和 String 对象

注意：用户可以使用 Applet Viewer 运行 DateReadAPP，但不能使用 Netscape 来运行它，因为要使对象系列化成为可能，需要对虚拟机做一些修改，而这些修改 Netscape 尚未实现。

35.3　写和读自己的对象

缺省情况下，用户可以写和读自己的大部分对象，正如对 Date 类所做的一样。但目前仍有一些限制（例如，如果对象引用一个内部（native）对等实体），然而对大部分由自己所创建的类，是可以系列化的。

程序清单 35.7 到程序清单 35.9 给出了系列化一个称为 SerializeObject 的示例类的源代码。

```
public class SerializeObject implements java.io.Serializable{
    public int first;
    public char second;
    public String third;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public SerializeObject (int first, char second, String third){
        this.first = first;
        this.second = second;
        this.third = third;
    }
}
```

程序清单 35.8 ObjWrite——写示例 SerializeObject 到文件

```
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import SerializeObject;
public class ObjWrite {
    public static void main (String args[]){
        try{
            FileOutputStream outputFile = new FileOutputStream("objFile");
            ObjectOutputStream serializeStream = new
ObjectOutputStream(outputFile);
            SerializeObject obj = new SerializeObject (1,'c',new String ("Hi!"));
            serializeStream.writeObject(obj);
            serializeStream.flush();
        } catch (Exception e) {
            System.out.println("Error during serialization");
        }
    }
}
```

程序清单 35.9 ObjRead——从文件读取相同的对象

```
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import SerializeObject;
public class ObjRead extends java.applet.Applet {
    public void init(){
        main(null);
    }

    public static void main (String args[]){
        SerializeObject obj;
        try{
            FileInputStream inputFile = new FileInputStream("objFile");
            ObjectInputStream serializeStream = new
ObjectInputStream(inputFile);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            obj = (SerializeObject)serializeStream.readObject();
        } catch (Exception e) {
            System.out.println("Error during serialization");
            return;
        }
        System.out.println("first is:" + obj.first);
        System.out.println("second is:" + obj.second);
        System.out.println("third is:" + obj.third);
    }
}
```

在前面示例类中，注意 SerializeObject 类引用了许多东西，其中包括一个类 String。正如所希望的，编译并运行这些类以后，输出结果如下：

```
First is:1
Second is:c
Third is:Hi!
```

在所有以上的这些代码中，令人最为惊讶的是传输对象是如此的容易。

35.4　定制对象系列化

有时候控制某个对象如何系列化非常有用，甚至非常必要。例如，如果用户要求对一个对象所拥有的数据以所有者形式加密，以便控制对该对象数据的访问，这种情况就不能使用缺省的系列化机制。

> **注意**：Special Serialization 在 JDK 1.2 中有一些修改。如果我们正在使用一个以前的 JDK 版本，此处所给出的方法将不起作用。相反，应参考 API 学习如何为自己的对象修改系列化。参阅 java.io.Replaceable 和 java.io.Resolvable 接口。

要重载对象系列化，必须使用签名在自己的类中定义两个方法：

```
private void writeObject(java.io.ObjectOutputStream out)
  thrwos IOException
private void readObject(java.io.ObjectInputStream in)
  throws IOException,ClassNotFoundException;
```

关于这一点用户可能要提的第一个问题是，如果 writeObject()和 readObject()不在 Serializable 接口，系列化系统是如何管理这些方法的调用的呢？答案是它使用了称为反射(Reflection)的一个东西。反射将在第 48 章"反射"中涉及，但其基本思想是允许程序基于它所知道的签名，访问组件的方法和构造器。反射一般是一个复杂的 API，对于用户的大部分程序，用户并不需要关心实际是如何使反射工作的。然而，用户应该知道，反射需要方法的准确签名。所以，正确使用这些签名是非常重要的。当方法的私有化失败时将导致系列化机制自动启用其缺省算法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用户的类不必去关心 super.writeObject()或 super.readObject()的调用,也不必去关心子类将如何使用系列化机制,在对象的每个部分被分别处理时对类进行系列化的。

另外,如果用户需要在 writeObject()方法中使用缺省的机制,调用 Out.defaultWriteObject()方法即可。或者从 readObject()方法调用 in.defaultReadObject()。

程序清单 35.10 包括一个称为 DateTest 的类,该类的三个单独的整型数——年(year)、月(month)和日(day),写出日期的值,并没有使用缺省的系列化机制。程序清单 35.11 和 35.12 包括了测试 DateTest 类的类。

```
import java.io.*;
import java.util.*;
import java.text.*;

public class DateTest implements Serializable{
  transient GregorianCalendar myDate;

  public void newDate(){
        myDate = new GregorianCalendar();
  }

  private void writeObject(ObjectOutputStream out) throws IOException{
        int year = myDate.get(Calendar.YEAR);
        int month = myDate.get(Calendar.MONTH);
        int day = myDate.get(Calendar.DAY_OF_MONTH);
        out.writeInt(year);
        out.writeInt(month);
        out.writeInt(day);
  }

  private void readObject(ObjectInputStream in) throws IOException, classNotFoundException{
        int year = in.readInt();
        int month = in.readInt();
        int day = in.readInt();
        myDate = new GregorianCalendar(year,month,day);
  }

  public String toString(){
        DateFormat df = DateFormat.getDateInstance();
        return "DateTest:" + df.format(myDate.getTime());
    }
  }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.io.*;

public class DateWriter{
  public static void main(String args[]){
        try{
        DateTest test = new DateTest();
        test.newDate();
        System.out.println("Writting test:" + test);
        FileOutputStream fout = new FileOutputStream("test.out");
        ObjectOutputStream oout = new ObjectOutputStream (fout);
        oout.writeObject(test);
        }catch (Exception ioe){
          ioe.printStackTrace(System.err);
        }
    }
}
```

```
import java.io.*;
public class DateReader{
  public static void main(String args[]){
        try{
        FileInputStream fin = new FileInputStream("test.out");
        ObjectInputStream oin = new ObjectInputStream (fin);
        DateTest test = (DateTest)oin.readObject();
        System.out.println("Read dateTest as " + test);
        }catch (Exception e){
                e.printStackTrace(System.err);
        }
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 36 章　远程方法调用

本章内容：

36.1　什么是远程方法调用
36.2　创建 Java 小程序客户程序
36.3　创建定制套接字
36.4　使用激活模型

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

36.1　什么是远程方法调用

客户/服务器模式优于其它系统，其特性之一是它的远程过程调用（Remote Procedure Calls，RPC）。从 JDK 1.1 开始，Java 也具有了与之类似的特性，称为远程方法调用（Remote Method Invocation），它允许在非客户端的计算机上创建、存放和处理对象。本章将介绍这一令人鼓舞并且功能强大的特性。

首先，我们有必要为远程方法调用（RMI）下一个定义。在对象系列化中，我们已经知道，用户可以将对象作为一个整体对待并沿着一个流传送对象。RMI 是对象系列化的姊妹概念，它允许用户沿着一个流传送一个对象，并且允许对象存放于不同的计算机，在这些计算机系统上激活方法。

换句话说，RMI 允许用户创建其方法可以被其它不同的计算机上的虚拟机（Virtual Machine）激活的 Java 对象。虽然这些对象驻留并处理于不同的计算机上，但它们在远程机器上的使用方式与一个本地对象非常相似。

36.1.1　创建远程对象

任何可以被远程调用的对象必须实现远程接口。但远程接口本身并不包含任何方法，所以很显然，只实现远程接口并没有完成所需要的工作。

为了得到一个有用的接口，我们必须为自己的对象创建一个新的接口，该接口必须扩展远程接口。新接口将包含所有可以远程调用的方法。正如用户已经猜想到的，远程对象也必须实现其自己的新接口。由于新的接口扩展了远程接口，实现了新的接口，就满足了远程对象对实现远程接口的要求，所实现的每个对象都将作为远程对象引用。所以要创建并实现远程对象（Remote Object）需要以下五个步骤：

1. 定义一个扩展远程接口的接口，该接口中的每一个方法必须声明它将产生一个 RemoteException 异常。

参见 12.2.1 中的“扩展其它接口”部分。

2. 定义一个实现该接口的类。因为新接口扩展了远程接口，满足了使新类成为一个远程对象的要求。该类必须提供一种方法，调度对于类实例的引用。

3. 使用 rmic 程序生成远程实现所需的主干和框架。

4. 创建一个客户程序对服务器进行 RMI 调用。

5. 启动 Registry（登记）并运行自己的远程服务器和客户程序。

> **注意：**当一个 RMI 方法需要参数时，所传送的对象使用对象系列化，正如第 35 章“对象系列化”中所讨论的一样。

36.1.2　一个 RMI 应用程序样本

为了理解 RMI，让我们来看一个完整的例子。同样，所使用的示例非常简单，它只创建并返回了一个字符串。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

36.1.3 创建远程接口

创建一个 RMI 应用程序的第一步是创建一个接口，该接口扩展远程接口。在这个接口中的每个方法都将可以被远程调用。如果事先有所考虑的话，也许我们已经认识到，本系统中使用这样一个接口正是面向对象的程序设计思想的完美体现。有了这一接口，调用远程对象的系统就可以像使用其它类一样使用该接口，而编译器并不需要知道有关方法体内部的任何代码。正如创建任何其它接口一样，必须确保每个方法的原型与将要在 final 类中使用的方法头完全匹配。程序清单 36.1 给出这一示例的一个简单远程接口。

```
public interface RemoteInterface extends java.rmi.Remote {
    String message (String message) throws java.rmi.RemoteException;
}
```

此处，我们使用一个单一的方法定义了一个接口。记住，远程接口并不真正有自己的方法，所以对于任何实现 RemoteInterface 的类，报文是其唯一需要定义的方法。

> **注意**：一个远程使用的接口可以使用任何类作为其参数或返回类型，只要那种类型实现 Serializable 即可。

36.1.4 创建实现类

第二步是定义一个类，实现新的远程接口 RemoteInterface。这个类的定义在程序清单 36.2 中给出。在这个例子中，message 只返回一个字符串，该字符串包含来自两方面的信息：传入的字符串和一个对于该对象是本地的串(name)。这样做可以帮助我们证明我们正在远程计算机上处理。为进一步强调这一点，可使用一个 println 语句，我们将会看到它将显示于远程计算机上，而不是客户机上。

```
import java.rmi.Naming;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
public class RemoteObject extends UnicastRemoteObject implements RemoteInterface{
  String name;
  public RemoteObject(String name) throws RemoteException{
    super();
    this.name = name;
  }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public String message(String message) throws RemoteException{
      String returnString = "My Name is:" + name + ",thanks for your message: " + message;
      System.out.println("Returning:" + returnString);
      return "My Name is:" + name + ",thanks for your message:" + message;
    }

    public static void main (String args[]){
      System.setSecurityManager (new RMISecurityManager());
      try{
        String myName = "ServerTest";
        RemoteObject theServer = new RemoteObject (myName);
        Naming.rebind(myName,theServer);
        System.out.println("Ready to continue");
      } catch (Exception e){
        System.out.println("An Exception occured while creating server");
      }
    }
  }
```

关于 RemoteObject 类有几个关键的地方应当注意。首先，RemoteObject 类扩展了 UnicastRemoteObject。在本章的范围内，我们可以将 UnicastRemoteObject 考虑为 RMI 服务器的 java.applet.Applet。我们也可以创建自己的 RemoteObject 类，但这不属于本章的范围。第二，服务器实现了在程序清单 36.1 中所定义的 RemoteInterface。

警告：在 JDK 1.02 下，我们需要导入并展开 java.rmi.UnicastRemoteObject，而不是 java.rmi.server.UnicastRemoteObject。所以在 JDK 1.02 下类的头为：

```
public class RemoteObject extends UnicastRemoteObject implements
RemoteInterface {
```

不幸的是，如果由于某种原因必须使用 JDK 1.02 VM 的话，这一变化会引起一些不兼容。

在 RemoteObject 中可以通过 RMI 调用的每个方法均必须声明它将产生一个 RemoteException 异常。注意，即使是构造器方法，也必须定义一个 RemoteException 异常。这样做的原因并不明显。在构造器方法中究竟是哪个命令可能产生一个异常呢？当然不是 name 的赋值语句，这样就剩下 super() 构造器调用了。可以肯定，UnicastRemoteObject 的构造器所做的事情是，通过在匿名端口(1099)侦听所进入的请求，导出远程对象(刚刚建立的那个对象)。不幸的是，如果用于通信的资源不可用，那么这种导出将会失败，引起一个异常，这就是我们的类必须抛出的那个异常。

> **注意：**与扩展其它类的所有类一样，缺省时 super()的调用是隐式的，为了帮助读者看清异常产生的地方，此处显式地给出。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

当然，RemoteObject 必须定义 RemoteInterface 的 message 方法，因为它实现了 RemoteInterface。由于该方法是我们试图使用 RMI 调用的方法，所以我们很关心这一方法。为使问题简单化，message 方法仅返回一个 String，其中包含它所接收到的信息。如果我们的客户程序接收到 String 返回，则可以肯定服务器收到了原始的 String。

main 方法所干的第一件事是建立一个新的 SecurityManager。这一安全管理器不必是 RMISecurityManager，但新的安全管理器必须允许加载 RMI 对象。确认 RMI 对象没有执行敏感操作是很重要的，缺省安全管理器不允许任何 RMI 对象被导出。

main 方法所做的第二件事是创建一个 RemoteObject 实例，该实例将是客户程序实际所"连接(attach)"的实例。该对象必须被绑定到 Registry。现在，关于如何完成这一绑定，应注意几个重要的问题。rebind()方法有两个参数，第一个参数是用来识别该对象的名字，第二个参数是对象本身。这里，对象与本地机绑定，并且不必完整地检查该名字。为使用完整合格的 URL，其语法格式为：

```
//host.name.com/bindname
```

正如在前面的例子中所看到的那样，只有绑定名是必须的。

> **注意**：使用 JDK 1.02 时，可以在对象的名字中使用空格，但现在不再支持这一特性。

警告：如果用户仍在使用 JDK 1.02，则需将安全管理器设为 java.rmi.server.StubSecurityManager，而不是 java.rmi.RMISecurityManager。所以需要将 main()方法的第一行改为：

```
System.setSecurityManager(new StubSecurityManager())
```

另外，用户还必须导入这个类，而不是 RMISecurityManager。

36.1.5 编译 RemoteServer

正像使用对象系列化一样，在编译 RemoteObject 时也必须包括附加类。

> **注意**：对于 JDK 1.02 用户，在编译 RemoteObject 之前，必须像在前一章中对象系列化时所详细介绍的那样，下载远程方法修补内容，并将 rmi.zip 文件添加到类路径中，如下所示：
>
> ```
> set classpath = c:\java\lib\classes.zip;c:\java\lib\rmi.zip;c:\objio.zip;.
> ```
>
> 从技术上讲此处包含 OBJIO.ZIP 文件是不必要的，但为了充分说明问题，将其放于此处也并非不是一个好注意。

现在可以编译 RemoteObject 了，输入如下：

```
javac RemoteObject.java
```

36.1.6 创建主干(Stub)

创建 RMI 的下一步是建立 RemoteObject 的主干和框架。要完成这一工作可使用 rmic 编译器，输入如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
rmic RemoteObject
```

正如所看到的，rmic 编译器的语法与 java 命令类似。事实上，在运行 rmic 时，它的许多可以使用的命令行选项与运行 java 命令时可使用的命令行选项是一样的。

> **注意**：不幸的是，在 JDK 1.1 下，JDK 的 Windows 版本存在一个小毛病，即在类路径中不能像在 java 或 javac 中一样，自动包含当前目录(.)。所以，如果用户所使用的是 JDK 1.1，则需要以如下所示的方式使用类路径选项(此处假设 JDK 1.1 安装在 C:\java 目录下)：
>
> ```
> rmic -classpath c:\java\lib\classes.zip;. RemoteObject
> ```
>
> 幸运的是，JDK 1.2 中不存在这一问题了。

rmic 编译器产生两个文件：

```
RemoteObject_Skel.class
RemoteObject_Stub.class
```

36.1.7　创建客户程序

创建 RMI 程序的下一步是创建客户程序，该客户程序真正地激活远程方法。程序清单 36.3 给出了一个示例类。

程序清单 36.3　RemoteObject.java——与 RemoteObject 类联系的客户程序示例

```
import java.rmi.RMISecurityManager;
import java.rmi.Naming;
public class RemoteClient {
  public static void main(String args[]){
    System.setSecurityManager(new RMISecurityManager());
    try{
      RemoteInterface server = (RemoteInterface)Naming.lookup("ServerTest");
      String serverString = server.message("Hello There");
      System.out.println("The server says : \n" + serverString);
    } catch (Exception e){
      System.out.println("Error while performing RMI");
    }
  }
}
```

RemoteClient 类最重要的部分是 try-catch 块中的两行：

```
RemoteInterface server = (RemoteInterface)Naming.lookup("Server Test");
String serverString = server.message("Hello There");
```

第一行代码在注册表(Registry)中查找定位一个称为“Server Test”的主干(如果回头查看

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下程序清单 36.2 中的第 21 行，就会发现我们使用该名字进行了绑定)。当程序创建了 RemoteInterface 实例以后，随后将使用字符串“Hello There”调用 message 方法。注意，这确实是一个方法调用，但是所激活的方法是在一个完全不同的系统上。该方法返回一个存储在 serverString 中的字符串，并随后打印出来。

正像对 RemoteObject 所做的那样，现在我们编译该客户程序：

```
javac RemoteClient.java
```

当然，此处假设已经为 RemoteObject 类设置了类路径(classpath)。

36.1.8 启动注册并运行代码

在实际运行 RemoteObject 类和 RemoteClient 类之前，用户必须首先在将要宿主 RemoteObject 的计算机上启动 RMI 注册(Registry)程序。即使将要运行 RemoteObject 的计算机与运行 RemoteClient 的是同一台机器(当只作为演示用时)，这一步也是必须的。为使注册程序工作，主干和框架文件的目录必须包含在为 rmiregistry 程序所设置的类路径中。

在 Windows 下启动注册程序，输入：

```
start rmiregistry
```

错误检查：如果用户输入 start rmiregistry 后，显示出“Bab Command or file name”错误，这并不是注册程序的原因，而是由于在路径中没有 Windows 的启动程序(start)的原因。start 程序一般存放在 Windows \ command 目录下，可将该目录增加到路径中或输入下述命令：

```
c: \ windows \ command \ start rmiregistry
```

注意：不幸的是对于 Windows 下的 JDK 1.1 用户，正如 rmic 一样，注册程序也没有在路径中包含当前目录，所以如果还没有升级为 JDK 1.2，启动注册程序时输入：

```
set classpath = c: \ java \ lib \ classes.zip;.
start rmiregistry
```

在大多数 UNIX 机器上，用户可以启动注册程序并将其置于后台，输入：

```
set classpath = /usr/java/lib/classes.zip;.
rmiregistry &
```

如果需要在框架和主干目录以外的其它目录启动注册程序，则应使用包含这些文件的目录代替类路径中的(.)。同样，应确保 classes.zip 的位置与所安装的位置一致。

注意：若用户仍在使用 JDK 1.02，则需要以不同的形式启动注册程序。在 JDK 1.02 下，下述命令可启动注册程序：

```
java java.rmi.registry.RegistryImpl
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

36.1.9　绑定 RemoteObject 到注册

当注册程序启动后，我们需要启动 RemoteObject 程序。此处启动 RemoteObject 与启动其它 Java 程序一样：

```
java RemoteObject
```

以后，我们也许需要将该对象推到后台，在 Windows 下，可输入下述命令在后台启动 Remote Object 程序：

```
javaw RemoteObject
```

在 UNIX 系统下，输入下述命令可在后台启动该对象：

```
java RemoteObject &
```

36.1.10　运行客户程序

最后一个任务是启动 RemoteClient。但在启动之前，应确保 RemoteObject 已经显示出如下信息：

```
Ready to continue
```

该信息暗示 RemoteObject 已经被导出并且绑定到注册上。要耐心一些，特别是在没有活动 Internet 连接的情况下，这将花费一定时间。如果在后台启动该对象，当然将永远看不到输出。

一旦 RemoteObject 给我们发出它已经准备好并且等待继续的通知，我们就可以启动 RemoteClient 了。这时，如果是在 Windows 下运行，则需要打开另一个 DOS 提示窗口，如果是在 UNIX 系统下，即使是将 RemoteObject 置于后台，也可能需要另一个会话（或 X 终端），以便能够看出服务器与客户端的差别。

最后，为运行 RemoteClient，输入：

```
java RemoteClient
```

屏幕上将出现下述输出结果：

```
The Server Says:
My Name is: ServerTest, thanks for your message: Hello There
```

注意服务器所产生和返回的字符串。如果查看一下 RemoteObject 窗口，可以看到输出结果为：

```
Ready to continue
Returning: My Name is: ServerTest, thanks for your message: Hello There
```

36.2　创建 Java 小程序客户程序

我们已经创建了一个使用 RemoteObject 的应用程序，现在来看看使用 Applet 如何建立一个客户程序，参见程序清单 36.4。可以看出，两种情况的差别并不很大。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意：由于 RMI 和对象系列化的需要而引起的虚拟机改变，使得直到在编写本书时，我们还只能在 Appletviewer 下运行这些 Java 小程序。Netscape 的 Navigator 和微软的 Internet Explorer 均不支持 RMI 和对象系列化。

程序清单 36.4 RemoteAppletClient.java——使用远程对象的一个 Java 小程序

```
/*
 *
 * RemoteAppletClient
 *
 */
import java.applet.Applet;
import java.rmi.RMISecurityManager;
import java.rmi.Naming;

public class RemoteAppletClient extends Applet {
  public void init(){
    System.setSecurityManager(new RMISecurityManager());
    try{
      RemoteInterface server = (RemoteInterface) Naming.lookup("ServerTest");
      String serverString = server.message("Hello There");
      System.out.println("The server says : \n" + serverString);
      } catch (Exception e){
        System.out.println("Error while performing RMI:" + e);
        }
    }
  }
```

36.3 创建定制套接字

有些情况下，我们可能需要改变一台机器与另一台机器之间的 RMI 通信方式。从第 30 章到第 34 章，我们探讨了套接字的用法，学习了关于 Java 固有的网络功能，这些功能中有一些是通过 RMI 缺省提供的，然而，缺省的实现可能并没有提供我们所需要的一些特性，如压缩或安全性。

现在我们来为 RMI 连接创建一个定制套接字实现。这需要如下三步骤：

1. 创建一个定制套接字，或选择一个套接字使用。
2. 使用新的套接字创建一个定制的 RMISocketFactory。
3. 在客户端和服务器设置套接字工厂。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

36.3.1　创建一个定制套接字

要建立一个定制套接字，一般来说需经过两步。第一步，创建一个定制流；第二步使用所创建的流创建一个套接字。如果用户实际使用的是一个来自第三方的套接字如 Sun 公司的 SSLSocket 类，可以跳过这一步。

创建定制流　使用一个定制 RMI 套接字的真正目的是为了提供一些附加功能，如压缩或安全性。如果没有可以提供附加功能的流，套接字本身并不会很好地工作，所以必须首先按照我们的目标定义一个流。实际上我们必须定义两个流：输入流 InputStream 和输出流 OutputStream。

在下面的例子中，我们将创建一组套接字，执行一组也许是最简单的安全性操作，我们将对输入和输出的每一个字节执行一个布尔非(NOT)操作。因为该操作非常简单，它只是一个方便的小算法。然而，由于 Java 没有字节操作的非(NOT)运算符，我们将使用异或操作符(XOR)将字节与 0xFF 异或，取得同样的结果。

现在，由于我们要看到最简单的例子，我们将扩展 java.io.InputStream。然而，在实际使用时可能要扩展更适合的流，如 BufferedInputstream 、FilteredInputStream 或 ObjectInputStream 等，这要根据特殊实现的主要用途而定。

程序清单 36.5 和程序清单 36.6 给出了新的流 CustomInputStream 和 CustomOutputStream。关于 Stream 类更详细的介绍参见第 27 章。

```
import java.io.*;
public class CustomInputStream extends FilterInputStream{

  public CustomInputStream(InputStream in) {
    super(in);
  }

  public int read() throws IOException {
    int code;
    try{
      code = in.read();
    } catch (EOFException e) {
      System.out.println("EOF");
      return -1;
    }
    //Don't invert a negative one, leave it as is.
    if (code != -1){
      //NOT the result
      code = (0xff^code);
      //mask off the top bits
      code = (0xff) & code;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }
      return code;
  }

  public int read(byte b[], int off, int len) throws IOException {
    int num = in.read(b,off,len);
    if (len <= 0) {
      return 0;
    }

    int i = 0;
    //convert the bytes
    for (;i<num;i++){
      b[off+i] = (byte)(b[off+i]^0xff);
    }
    return num;
  }
}
```

```
import java.io.*;
public class CustomOutputStream extends FilterOutputStream {
  public CustomOutputStream (OutputStream out){
    super(out);
  }

  public void write(int b) throws IOException{
    out.write(b ^ 0xff);
    out.flush();
  }

  public void write (byte b[],int off, int len) throws IOException{
    for (int i = 0; i < len; i++) {
      byte b2 = b[off + i];
      write(b2);//[off + i]);
    }
  }
}
```

创建定制套接字 现在我们已经有了一个新的流,可以创建新的套接字类了。新的套接

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

字将使用新流，但这与标准的套接字类并没有太大差别。程序清单 36.7 给出了这种实现。注意在 getInputStream()和 getOutputStream()方法中，返回的是新的流类而且不是原始的流类(super.getInputStream()或 super.getOutputStream())。

```
import java.io.*;
import java.net.*;
public class CustomSocket extends Socket {
  //The InputStream of the socket
  private InputStream in;
  //The output stream for the socket.
  private OutputStream out;

  //null constructor, since you want to support both types of ocket constructors
  //you need to prototype both of them.
  public CustomSocket() {
    super();
  }

  public CustomSocket(String host, int port) throws IOException {
    super(host, port);
  }


  public InputStream getInputStream() throws IOException {
    if (in == null) {
      //create a new stream from the normal socket stream
      in = new CustomInputStream(super.getInputStream());
    }
    return in;
  }

  public OutputStream getOutputStream() throws IOException{
    if (out == null) {
      //create a new stream from the normal socket stream
      out = new CustomOutputStream(super.getOutputStream());}
    return out;
  }
}
```

创建定制 ServerSocket　RMI 系统必须能够侦听到新的连接(ServerSockets)。显然，我们要使这些连接具有与刚刚创建的 Socket 类相同的类。所以，我们也必须建立一个定制的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

ServerSocket,该 ServerSocket 将使用 CustomSocket 类。为此,新类实际上只要替换 accept()方法,并使用程序清单 36.7 的 CustomSocket 类即可。程序清单 36.8 给出了新的 ServerSocket 查找 CustomSocket 实现的代码。

程序清单 36.8 CustomServerSocket.java——扩展 ServerSocket 并使用 CustomSocket 类

```
import java.io.*;
import java.net.*;
public class CustomServerSocket extends ServerSocket {
    public CustomServerSocket(int port) throws IOException {
      super(port);
    }

  public Socket accept() throws IOException {
        Socket s = new CustomSocket();
        implAccept(s);
        return s;
    }
}
```

36.3.2 创建定制的 RMISocketFactory

现在我们已经创建了自己的 Socket 类,我们可以使用该套接字类创建一个 RMISocketFactory。RMISocketFactory 实际上提供了将要在 RMI 应用程序中使用的套接字。RMISocketFactory 有两个抽象方法:createSocket()和 createServerSocket(),新的套接字工厂必须为其提供实现。显然此处的目的是让这两个方法返回 CustomSocket 或 CustomServerSocket。在程序清单 36.9 中我们将看到这一点。

注意:在阅读程序清单 36.9 时,应注意程序中实际上有四个方法。对我们来说后两个方法才是有意义的(创建了特殊类型的套接字),在程序清单 36.9 中这两个方法被重载以提供 CustomSocket 和 CustomServerSocket。让我们感到不可理解的是其前两个方法。这两个方法必须返回缺省的套接字。为什么要这样做呢?答案是当 RMI 执行其初始的注册系统"查找"时,RMI 系统仍然需要能够获得"清洁"的套接字。我们知道,注册(Registry)系统并不关心远程对象与客户是如何交谈的,它只希望我们以一种标准的套接字与之交谈。

注册系统实际上只负责让这些对象相互知道对方所处的位置。在这之后对象之间就可以直接相互交谈,而无需注册系统处于中间。为完成这一工作,如果指定的套接字类型未知的话,我们就仍然必然提供一个缺省的套接字实现。

程序清单 36.9 CustomRMISocketFactory——扩展 RMISocketFactory 类并使用定制套接字

```
import java.io.*;
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
import java.net.*;
import java.rmi.server.*;
public class CustomRMISocketFactory extends RMISocketFactory {

    private RMISocketFactory defaultFactory = RMISocketFactory.getDefaultSocketFactory();

    public Socket createSocket(String host, int port) throws IOException {
      return defaultFactory.createSocket(host, port);
    }

    public ServerSocket createServerSocket(int port) throws IOException {
            return defaultFactory.createServerSocket(port);
    }

  public Socket createSocket(String host, int port,SocketType type)    throws IOException    {
    String protocol = type.getProtocol();
    if(protocol.equals("custom"))
        // Use default bit pattern for the XorSocket.
        return new CustomSocket(host, port);
    return createSocket(host,port);
  }

  public ServerSocket createServerSocket(int port,SocketType type)throws IOException    {
    String protocol = type.getProtocol();
    if(protocol.equals("custom"))
      return new CustomServerSocket(port);
    return createServerSocket(port);
  }
}
```

36.3.3　在应用程序中指定套接字工厂

创建定制套接字过程的最后一步是，实际告诉应用程序开始使用新的套接字工厂。其中唯一应注意的是必须确保同时对客户和服务器组件进行修改。如果未能这样做，显然将以其中一端为杂乱数据而结束。

要改变套接字工厂，我们所需要做的就是调用 setSocketFactory 方法。setSocketFactory() 是 RMISocketFactory 的一个静态方法。所以我们需要添加的代码如下所示：

```
RMISocketFactory.setSocketFactory(new CustomRMISocketFactory());
```

在新的客户和服务器程序的 main 方法中，setSocketFactory()方法一般应为第一行。程序清单 36.10 和 36.11 给出了来自程序清单 36.2 和 36.3 的 HelloWorld 应用程序。注意唯一的变化是所插入的行(现在需要导入 RMISocketFactory)。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

必须进行的第二个改动是为 UnicastRemoteObject 调用构造器。该构造器的调用可以不带任何参数(与未使用定制套接字时的一般调用一样),也可以指定端口和套接字类型。

在指定参数的情况下,我们可以指定端口为 0(即使用缺省的端口),并且必须提供一个 SocketType 对象。SocketType 将告诉 RMISocketFactory 我们所需使用的套接字类型。SocketType 构造器如下所示:

```
public SocketType(String protocol, byet[] refData, Object serverData)
```

sefData 可提供一些附加的协议数据,serverData 可提供附加服务器协议信息。但在此处,我们不必担心这些。我们所关心的唯一一件事是协议的名字,即本章中我们选择使用的"custom"。现在,为能够实际调用 UnicastRemoteObject 的构造器,我们需要修改 RemoteObject 的构造器,如下所示:

```
public RemoteObject(String name) throws RemoteException{
  super(0,new SocketType("custom",null,null));
  this.name = name;
}
```

在程序清单 36.10 和 36.11 中,我们将看到这些改动的完整清单。当编译完程序清单 36.10 和 36.11 后,可以像本章前面使用普通套接字实现时一样,运行该应用程序。但是,在启动之前不要忘了编译其主干。

[illegible]

```
import java.rmi.server.SocketType;

public class RemoteObject extends UnicastRemoteObject implements RemoteInterface{
  String name;
  public RemoteObject(String name) throws RemoteException{
    super(0, new SocketType("custom", null, null));
    this.name = name;
    }

    public String message(String message) throws RemoteException{
      String returnString = "My Name is:" + name + ",thanks for your message:" + message;
    System.out.println("Returning:" + returnString);
    return "My Name is:" + name + ",thanks for your message:" + message;
  }

  public static void main (String args[]){
    try{
      RMISocketFactory.setSocketFactory(new CustomRMISocketFactory());
      System.setSecurityManager (new RMISecurityManager());
      //RMISocketFactory.setSocketFactory(new XorSocketFactory());
      //CustomRMISocketFactory());
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        String myName = "ServerTest";
        RemoteObject theServer = new RemoteObject (myName);
        Naming.rebind("/" + myName, theServer);
        System.out.println("Ready to continue");
      } catch (Exception e){
        System.out.println("An Exception occurred while creating server");
        e.printStackTrace(System.out);
      }
    }
  }
```

```
import java.rmi.RMISecurityManager;
import java.rmi.server.RMISocketFactory;
import java.rmi.Naming;
public class RemoteClient {
  public static void main(String args[]){
    try{
      System.setSecurityManager(new RMISecurityManager());
      RMISocketFactory.setSocketFactory(new CustomRMISocketFactory());

      RemoteInterface server = (RemoteInterface) Naming.lookup("ServerTest");
      String serverString = server.message("Hello There");
      System.out.println("The server says : \n" + serverString);
    } catch (Exception e){
      System.out.println("Error while performing RMI:" + e);
      e.printStackTrace(System.out);
    }
  }
}
```

36.4　使用激活模型

在 RMI 实现中，另一个新的特性是远程激活一个对象的功能。在使用激活器(Activator)之前 RMI 存在的问题是，我们必须启动对象并将其注册到系统中，正如本章所做的那样。然而，这一解决方案并不十分理想。

如果远程对象由于某种原因而死掉(由于异常的错误)，或如果远程对象认为它正在进行处理(例如，如果一台机器没有重新设置)，则当我们的远程应用程序试图使用该远程对象时，将是不可用的。也有可能会由于某种原因，使一个远程对象的启动完全失败。

通过使注册系统(Registry)使用一个 Activator(激活器)，它的激活功能可帮助我们解决这一问题。激活器(Activator)将检查远程对象的状态，并执行使它重新运行所必须的初始化。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

36.4.1 创建可激活对象

创建一组可以被激活的远程对象时，第一步是从 java.rmi.activation.activatable 扩展每一个远程对象，而并非从本章已经看到的 UnicastRemoteObject 进行扩展。

由于我们的对象现在扩展为可激活的(Activatable)，所以还必须修改其构造器签名。

```
public MyRemoteInterfaceImpl(ActivationID id, MarshalledObject data)
throws RemoteException{
//使用激活系统注册对象
//然后在一个匿名端口导出
super(id,0);
}
```

然而，主要的改动是在新的 main 方法中。激活器的设置需要执行一些附加操作。

第一个任务是定义可激活对象所存放的位置，当需要时该 URL 将在远程守护线程创建对象时使用。

```
java.net.URL location = new java.net.URL("file:/src/se4/");
```

当我们知道了主干和框架文件所放置的位置后，下一步需要做的是定义一个 CodeSource 对象。CodeSource 构造器需要两个参数。第一个参数是上面所定义的 URL，第二个参数是公共密钥。此处，我们不必定义所需的公共密钥，其设为 null。

```
CodeSource source = new CodeSource(location,null);
```

下一步是获得系统属性，并创建一个 ActivationGroupID。ActivationGroupID 用于创建一个 ActivationDesc。现在必须使用 ActivationDesc 实际注册新对象。

```
Properties props = (Properties)System.getProperties().clone();
ActivationGroupID agid = ActivationGroup.getSystem().registerGroup(
    new ActivationGroupDesc(props);
MarshalledObject data = null;
ActivationDesc desc = new ActivationDesc
    (agid,"RemoteActivatableObject",source,data);
RemoteInterface remoteInterface = (RemoteInterface)
    Activatable.register(desc);
```

现在，我们有了一个可以重新绑定的远程接口对象，正如我们在前面几节所做的那样。然而，一旦我们绑定该对象，就不必使系统继续等待连接，我们可以退出系统。

```
Naming.rebind("ServerTest",remoteInterface);
System.out.println("Exported interface, ready to go");

//现在程序退出，rmid 从此处接管
System.exit(0);
```

注意：在运行本节中例子时要确保所使用的 RemoteClient 来自于程序清单 36.3，而不是来自于程序清单 36.11。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 36.12 给出了新的 RemoteActivatableObject。为建立一个新的程序，必须首先编译 RemoteActivatableObject.java，然后对其运行 rmic。

```
import java.rmi.Naming;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.RMISocketFactory;
import java.rmi.activation.*;
import java.rmi.server.SocketType;
import java.rmi.MarshalledObject;
import java.security.CodeSource;
import java.util.Properties;


public class RemoteActivatableObject extends Activatable implements RemoteInterface{

  public RemoteActivatableObject(ActivationID id, MarshalledObject data)
    throws RemoteException {
    super(id, 0);
  }

  public String message(String message) throws RemoteException{
    String returnString = "Thanks for your message:" + message;
    System.out.println("Returning:" + returnString);
    return "Thanks for your message:" + message;
  }

  public static void main(String[] args) throws Exception {
    System.setSecurityManager(new RMISecurityManager());

    java.net.URL location = new java.net.URL("file:/src/se4/");
    CodeSource source = new CodeSource(location, null);
    Properties props = (Properties)System.getProperties().clone();
    ActivationGroupID agid = ActivationGroup.getSystem().registerGroup(
            new ActivationGroupDesc(props));
    MarshalledObject data = null;
    ActivationDesc desc = new ActivationDesc (agid, "RemoteActivatableObject", source, data);
    RemoteInterface remoteInterface = (RemoteInterface)Activatable.register(desc);
    Naming.rebind("ServerTest", remoteInterface);
    System.out.println("Exported interface, ready to go");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        //Now exit the program, the rmid will take over from here
        System.exit(0);
    }
}
```

然后，当启动 rmiregistry 后，再启动 rmid，rmid 将在需要时实例化远程对象。最后，我们可以像前面一样，运行 RemoteActivatableObject 和 RemoteClient。简单地说，可以使用下列过程建立程序：

```
javac RemoteActivatableObject.java
rmic RemoteActivatableObject
start rmiregistry
rmid
```

现在 rmid 将持续地运行，就像前面的 RemoteObject 一样，所以我们现在需要另外一个终端窗口(或 DOS 提示窗口)。在此我们可以启动可激活对象并运行客户程序：

```
java RemoteActivatableObject
java RemoteClient
```

运行结果如下所示：

```
G: \ src \ se4 > java  - Djava.rmi.server.codebase = file:/src/se4/
     RemoteActivatableObject
exported interface, ready to go

G: \ src \ se4 > java RemoteClient
The server says :
Thanks for your message:Hello There
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 37 章　管理 API

本章内容：

37.1　JMAPI 组件

37.2　JMAPI 小程序

37.3　创建管理对象

37.4　Admin View Module

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

37.1 JMAPI 组件

Java Management API(JMAPI)是一个比较全面的子系统,它主要用于创建中心管理工具可管理的 Java 对象,也可以用来创建管理工具本身。

在 Java 管理框架内,有三个主要组件:

管理对象服务器(Managed Object Server)

任何数量的管理"设备(appliance)"

提供管理设备(从烤箱(toaster)到大型计算机)用户界面的一个 JMAPI 小程序

管理对象服务器(Managed Object Server)包括一个管理对象工厂(Managed Object Factory),用于创建管理对象。虽然 JMAPI 系统是管理设备的,但大多数的管理逻辑却驻留在这些管理对象之中。

管理对象与通常存在于设备中的代理对象进行通信。这些代理对象实现管理对象的请求。代理对象有时需要本地(native)库来控制应用。JMAPI 包括对内部库的特殊支持,允许代理为其正在运行的任何事件平台卸载本地库(假设本地库正是为该平台而存在)。Sun 公司使用"设备(appliance)"这一术语,描述 JMAPI 所管理的设备——虽然目前所处理的设备大多是计算机,但将来有可能使用 JMAPI 来管理日常的家用电器。

JMAPI 小程序所提供的是管理用户界面。网络管理应用程序通常是很复杂的,需要许多提供数据和接收输入的方法。Admin View Module(AVM)包括了许多这样的组件。JMAPI 小程序通过管理对象接口与管理对象进行通信,管理对象接口使用 RMI 传输信息。

37.2 JMAPI 小程序

JMAPI 小程序表示的是 Java Management API 的用户接口。一个典型 Java 小程序可以使一个用户查看到可用的管理设备,并修改这些设备的各种设置。与普通的 Web Java 小程序不同,JMAPI 小程序需要一些特殊的 Java 小程序参数,以便与管理对象服务器进行通信。JMAPI 提供了三种不同的启动 Web 页面,以发射其它的 JMAPI 小程序:

- JmapiHome.html
- MOContentManagerApplet.html
- MOProperBookApplet.html

37.2.1 JmapiHome.html

JmapiHome.html 页面包含一个发射器,它可以启动其它的管理小程序。发射器可用一个可选页面注册对象,以限制可使用的小程序。缺省情况下,发射器显示所有的可用页面。JmapiHome.html 有几个配置参数,使用 < PARAM > 标签可指定这些参数。这些参数的名字及其含义如下:

- host 管理对象服务器所运行的主机名(必选)
- port 管理对象服务器的端口号(必选)
- domainName 本小程序正在管理的管理域的名字(可选)
- contexName 一个持续上下文的名字,用于初始化本地管理上下文(可选)

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

- pageRegistry 页面注册对象的名字，包括加载器所显示的信息

37.2.2　MOContentManagerApplet.html

MOContentManagerApplet.html 页面显示一个管理对象的类内容管理器。每一种类型的管理对象可以有一个表示对象，用于管理本类型的对象。为了管理一个 MOContentManagerApplet 对象，对象的类型必须定义一个表示对象。在 MOContentManagerApplet 的 < PARAM > 标签中，指定 ManagedObjectClassName 参数以通知所需要管理的对象类型。

37.2.3　MOPropertyBookApplet.html

MOPropertyBookApplet.html 显示一个对象的属性册，该小程序对所需要管理的每种对象，也要求定义一个表示对象。

37.2.4　访问管理对象

MOFactory 类提供对运行于管理对象服务器的管理对象工厂的访问。使用该类可以创建、修改及删除管理对象。也可以枚举一个对象集，查询满足指定条件的对象。

在执行任何管理功能之前，必须在 MOFactory 中使用 initialize 方法，指定管理对象服务器的主机名和端口号：

```
MOFactory.initialize("mgmt_server",1234);
```

要创建一个管理对象的新实例，可使用该类的全名调用 newObj 方法：

```
MyManagedObject.obj = (MyManagedObject)MOFactory.newObj(
    "demopkg.mgmt.MyManagedObject");
```

对管理对象的 get 和 set 方法，需要一个安全性上下文来执行操作。可以传送 null 作为安全性上下文；但对一个生产系统，应该有一个适当的安全性上下文：

```
SecurityContext context = null;
obj.setSomething(context,"somethingValue");
obj.setSomethingElse(context,"otherValue");
```

当创建一个对象后，要使用 addObject 方法将其添加到管理服务器：

```
obj.addOject(context);
```

modifyObject 和 deleteObject 方法使用管理对象服务器登记对象的改变和对象的删除：

```
obj.modifyObject(context);
obj.deleteObject(context);
```

listMOClass 方法返回一个向量，其中包含某一特殊类的所有管理对象：

```
Vector V = MOFactory.listMOClass("demopkg.mgmt.MyManagedObject");
```

用户可以指定一个可选的查询对象，搜索具有某些属性的管理对象。QueryExp 类包含创建查询表达式的方法。可以将查询表达式传送到 ListMOClass 方法，以限制搜索。这样做可以节省大量的时间。如果服务器上有大量的管理对象，当仅需要一个很小的子集时，我们并不需要服务器将所有的对象都发送给我们的客户程序。较好的方法是使服务器清除那些不需要

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的对象,而只发送那些我们所需要的对象。

37.3 创建管理对象

在使用 JMAPI 的大部分时间里,用户所使用的管理对象是现有的管理对象。然而有的时候用户也需要创建管理对象,这种对象对 JMAPI 小程序是可见的。

37.3.1 管理对象中定义属性

要建立一个对 JMAPI 小程序可见的属性,只要提供 get 和 set 方法即可,正如在 JavaBeans API 中一样。JMAPI 的 get 和 set 方法与 Beans 的 get 和 set 方法之间的区别之一是,JMAPI 的方法使用了一个附加的 SecurityContext 参数。程序清单 37.1 中所给出的类定义使用称为"address"的属性创建了一个管理对象。

```
public class SimpleAddressMOImpl extends LogicalElementMOImpl
    implements SimpleAddressMOImpl
{
    PrismVar String address;

    public String getName(SecurityContext context)
    {
        return (String) getAttribute("address");
    }

    public void setName(SecurityContext context, String hostName)
    {
        setKnownAttribute("address", hostName);
    }
}
```

注意:JMAPI 中所定义的管理对象的文件名不是使用一般的 .java 作为扩展名,而是使用 .mo。

管理对象文件中的 PrismVar 关键字所表示的是管理对象的属性。另外,使用 setKnownAttribute 可永久性地设置属性。换句话说,当设置了一个已知属性时,设置值被保存到数据库中,当下次实例化该对象时,将获得该属性值。

37.3.2 管理对象中定义方法

管理对象的实现是 RMI 服务器,在对象文件 .mo 中所定义的任何方法均在 RMI 中有定

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

义,并且可以从管理小程序中调用。用户仅需要提供方法的实现即可,其余的部分由 JMAPI 框架负责。

37.3.3　编译管理对象

由于管理对象定义在 .mo 文件中,所以需要有一种方法把这种文件转换为一组 Java 类。对此我们可以使用 Moco 程序(Managed Object Compiler)。要编译一个管理对象,只要输入如下命令即可:

```
Java moco.Moco YourManagedObject.mo
```

管理对象编译器为每个管理对象创建三个 Java 源文件:接口定义、实现及数据库操作。对于一个名为 SimpleAddressMO.mo 的文件,所产生的 Java 源文件为:

```
SimpleAddressMO.java
SimpleAddressMOImpl.java
DBSimpleAddressMO.java
```

警告:如果我们所定义的管理对象引用了其它管理对象,必须在 Moco 命令行中包括这些 .mo 文件。

37.3.4　导入管理对象

我们在 JMAPI 系统中使用管理对象之前,必须首先将其导入 JMAPI 数据库中。importclasses 命令可将一个或多个类文件导入到 JMAPI 数据库系统中。如果我们所定义的类引用到其它尚未存放在数据库中的管理对象,则必须同时将所有这些对象导入数据库。下面的命令将 SimpleAddressMO 导入 JMAPI 数据库:

```
importclasses SimpleAddressMO
```

当管理对象装载到系统之后,JMAPI 小程序便可以访问了。

37.4　Admin View Module

虽然我们偶尔也需要创建新的管理对象,但更多的情况是创建 JMAPI 小程序。毕竟只有那么多新类型的对象需要管理,而显示和操作对象的新方法却总是存在。

由于管理应用程序需要更复杂的用户界面,JMAPI 提供了一组很有用的组件,称为 Admin View Module(AVM)。虽然这些组件最初的目的是用于 JMAPI 应用程序,但它们并不依赖于 JMAPI 的任何其它部分。换句话说,我们可以在任何所需要的应用程序中使用 AVM,而无需依赖于 JMAPI 的其余部分。

37.4.1　内容管理器和可选择对象

在典型的管理应用程序中,都有一组可以选择并执行操作的项。Java AWT 本身并不提供这些功能,而是将其留给了用户,或此处的 AVM。

AVM 定义了一个 ContentManager 类,该类是一个抽象类,用于显示一组对象,允许我们

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

选择其中的一些对象并对其执行命令。创建一个内容管理器是一项比较复杂的工作，但是由于 AVM 提供了几个有用的内容管理器，我们也许根本不需要再创建。

ContentManager 类使用 Selectable 接口通知对象其选择或未选择的具体情况，并找出每个对象所支持的命令。每个对象都可以支持自己的命令，当对象被选择后，这些命令可在一个菜单上获得。Selectable 接口包括如下一些方法：

```
public void select()
public void deselect()
public boolean isSelected()
public void toggleSelection()
public Command[] getCommands(String commandType)
public Command[] setCommands(String commandType, Command[] commands)
```

内容管理器和其它对象使用 select 、deselect 、toggleSelection 以及 isSelected 方法，以控制是否选择一个对象。如果对象根据其选择情况需要改变其外形，可在这些方法的实现中来完成。

getCommands 和 setCommands 方法控制对象所支持的 Command 对象数组。当选择一个对象后，其命令出现在一个菜单上。当从菜单中选择一个命令后，执行该命令的 execute 方法以完成所需操作。

除了上述的选择和命令方法之外，selectable 接口中还定义了三个字符串常量：SELECTED_COMMANDS、CREATE_COMMANDS 及 VIEW_COMMANDS。当我们在一个可选择的接口中获取或设置命令时，必须指出所需要的命令是用于创建、观察还是选择菜单。内容管理器使用这些命令类型，并维护这三个菜单。

Command 接口非常简单，它只包含三个方法：

```
public void execute(Object executor)
public String getLabel()
public String getName()
```

作为菜单选择的结果，调用 execute 方法并执行所需的操作。getLabel 方法返回菜单中所显示的字符串。getName 方法返回所指定的命令名。这使我们在根据环境改变标签(可视部分)的同时，可以保持名字的一个内部版本。

程序清单 37.2 给出一个使用 SimpleContentManager 类演示如何设置一个命令的小程序。

```
import sunw.admin.avm.base.*;
import java.applet.*;
import java.awt.*;
import java.util.Vector;
import java.net.URL;

public class SimpleCMApplet extends Applet
{
    Frame currFrame;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    static String[] playmateInfo = {
        "Katy", "Sammy", "Bunnie"
    };

// For the play command, just print a dialog

    class PlayCommand implements Command {
        public String getLabel() { return "Play"; };
        public String getName() { return "PlayCommand"; };

        public void execute(Object executor) {
            play(executor);
        }
    };

    public void init()
    {

// Load an image to display for the icon part of the simple content manager

        URL imgURL = null;
        try {
            imgURL = new URL(getDocumentBase(), "katyface.GIF");
        } catch (Exception ignore) {
        }

        Image img = getImage(imgURL);

        MediaTracker tracker = new MediaTracker(this);
        try {
            tracker.addImage(img, 0);
            tracker.waitForAll();
        } catch (Exception e) {
            e.printStackTrace();
        }

// Create a simple content manager

        SimpleContentManager cm = new SimpleContentManager(
            "Play Manager", img);

// Create the data for the content manager's table
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        TableData data = new TableData(playmateInfo, "|");

        cm.setTableData(data);

// Create an array of column names
        String[] cols = new String[1];
        cols[0] = "Name";

// Set up a filter to display the column names
        cm.setFilterPipe(new TableFilterPipe(data, cols));

// Create an empty query space
        cm.setQuerySpace(new QuerySpace());

// Set up the default sort pipe
        cm.setSortPipe(new TableSortPipe(cm.getFilterPipe(), cols));

// Cerate an array of column widths
        int[] widths = new int[1];
        widths[0] = 20;

// Set up the view properties pipe, which sets the column widths
        cm.setViewPipe(new TableViewPropertiesPipe(cm.getSortPipe(),cols, widths));

// Make the Content Manager look at the view pipe to get the column data
        cm.getViewPipe().addObserver(cm);

// Create an array of commands
        Command commands[] = new Command[1];
        commands[0] = new PlayCommand();

// Assign the commands to the content manager
        cm.setCommands(Selectable.SELECTED_COMMANDS,commands);

// Make the content manager process the data
        data.changed();

        setLayout(new GridLayout(1, 0));

// Create a top-level content manager to contain the simple manager
        TopLevelContentManager topCM = new TopLevelContentManager();

        add(topCM);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// Add the simple manager to the top manager
        topCM.add("Center", cm);

// Make the top-level manager receive events from the simple manager
        cm.addItemListener(topCM);

// Activate the simple manager (make it the current panel)
        topCM.select(cm);

// The command classes need a reference to this applet's frame
        currFrame = Util.findFrame(this);


    protected void play(Object executor)
    {
        if (! (executor instanceof SimpleContentManager)) return;

// Get a reference to the current content manager
        SimpleContentManager cm = (SimpleContentManager) executor;

// Get the data table
        Table table = cm.getTable();

// Find the selected row
        int index = table.getSelectedIndex();

// Get the first column in the row
        String whichSelection = (String) table.getItem(index, 0);
// Just put up a dialog showing who the playmate is
        InformationDialog info = new InformationDialog(currFrame, "Playing with "+ whichSelection);
        info.setVisible(true);
    }
}
```

在一个典型的管理应用程序中，命令对象将激活管理对象的方法。由于内容管理器与 JMAPI 框架的其它部分没有联系，命令对象可以执行我们所需要的任何功能。例如，如果用户在一个航空公司工作，也许会创建一个内容管理器，使用“create flight plan”、“cancel”或“divert”命令显示一个航班表。

37.4.2　图标

当建立一个复杂的用户界面时，我们常常需要使用图标图像化表示各个项。虽然可以创建 AWT canvas 类的一个子类，但我们很快会发现这将需要自己编写大量的代码。AVM 中的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Icon 类实现了 Selectable 接口，并允许由内容管理器来管理。IconCanvas 类管理多个图标的显示及可能的选择。

为创建一个 Icon 对象，只要提供一个名字和一个图像或 AWT 组件即可。下述代码行创建一个 Icon：

```
Icon myIcon = new Icon(myImage,"My Image");
```

IconCanvas 对象实际上是 Panel 的一个子类而不是 Canvas 的子类。要将图标放置到画布上，只要调用 add 方法即可，就像使用其它的面板对象一样。用户还可以为 IconCanvas 设置一个布局管理器，确定图标所放置的位置。然而，不允许将除图标以外的对象添加到画布上。

程序清单 37.3 给出了一个创建三个图标并显示的样本小程序。

```
import sunw.admin.avm.base.*;
import java.applet.*;
import java.awt.*;
import java.net.URL;

public class IconApplet extends Applet
{

    public void init()
    {

// Load images to display in applet

        URL imgURL = null;
        try {
            imgURL = new URL(getDocumentBase(), "moeicon.gif");
        } catch (Exception ignore) {
        }

        Image moeImg = getImage(imgURL);

        try {
            imgURL = new URL(getDocumentBase(), "larryicon.gif");
        } catch (Exception ignore) {
        }

        Image larryImg = getImage(imgURL);

        try {
            imgURL = new URL(getDocumentBase(), "curlyicon.gif");
        } catch (Exception ignore) {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
        Image curlyImg = getImage(imgURL);

        MediaTracker tracker = new MediaTracker(this);
        try {
            tracker.addImage(moeImg, 0);
            tracker.addImage(larryImg, 0);
            tracker.addImage(curlyImg, 0);
            tracker.waitForAll();
        } catch (Exception e) {
            e.printStackTrace();
        }

// Create a canvas for the icons

        IconCanvas canvas = new IconCanvas();
        canvas.setLayout(new FlowLayout());

// Create the icons
        Icon moeIcon = new Icon(moeImg, "Moe");
        Icon larryIcon = new Icon(larryImg, "Larry");
        Icon curlyIcon = new Icon(curlyImg, "Curly");

// Add the icons to the canvas
        canvas.add(moeIcon);
        canvas.add(larryIcon);
        canvas.add(curlyIcon);

// Add the canvas to the applet
        setLayout(new GridLayout(1, 0));
        add(canvas);
    }
}
```

图 37.1 显示了 IconApplet 活动时状态。注意 Icon 类同时显示了图像和标签。



图 37.1　Icon 类使有标签并且可选择的图像显示变得容易了

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

37.4.3 属性册

属性册是分隔信息成为逻辑分隔"页"的一种方法。通常,一个属性册与一个特殊的对象联系在一起,其中包含了关于该对象的配置信息。然而,一个属性册无需与某一特殊项建立联系。属性册类似于一个标签面板,我们可能在 Java Foundation Classes 中已经看到过。

属性册的左侧是一个列,列出了各种可以使用的页。这个列称为索引,是由 PropertyBookIndexPanel 类实现的。

属性册的底部包含应用于属性册各页的按钮。一般地,可以有 Apply(应用)、Cancel(取消)及 Default(缺省)等按钮。PropertyBookButtonsPanel 控制这些按钮,其中除了按钮之外不包含其它东西。这就形成了属性册较为标准的用户界面。

属性册的顶部包含一个菜单条,并为"标识面板"留有空间,标识面板包含有关属性册的信息。

属性册的主体显示多个属性册节中的一个,每个节是 PropertyBookSection 类的一个实例,它是面板(Panel)的基础,其中还包括两个专用于属性的方法:apply 和 reset。如何实现这两个方法由用户决定。

在简单的情况下,用户只要为所需的每个不同的节创建 PropertyBookSection 的子类,然后将所创建的节添加到属性册即可。程序清单 37.4 给出了一个非常简单并且完全没有用的属性册小程序。

```
import sunw.admin.avm.base.*;
import java.applet.*;
import java.awt.*;

public class PropertyApplet extends Applet
{
    public class InfoSection extends PropertySection
    {
        String who;

        public InfoSection(String who)
        {
            this.who = who;

            setLayout(new BorderLayout());

            add("Center",new Label("Here is the information about " + who));
        }
    }

    public void init()
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        PropertyBook book = new PropertyBook();

        PropertySection moeSection = new InfoSection("Moe");
        book.addSection("Moe", moeSection);

        PropertySection larrySection = new InfoSection("Larry");
        book.addSection("Larry", larrySection);

        PropertySection curlySection = new InfoSection("Curly");
        book.addSection("Curly", curlySection);

        PropertySection shempSection = new InfoSection("Shemp");
        book.addSection("Shemp", shempSection);

        PropertySection curlyJoeSection = new InfoSection("Curly Joe");
        book.addSection("Curly Joe", curlyJoeSection);

        setLayout(new GridLayout(1, 0));
        add(book);
    }
}
```

37.4.4　任务页

任务页有些类似于属性册。任务页由使用不同信息的多个面板组成。最大的差别是，任务页代表了一个步骤序列。当安装新软件时常常遇到这种界面。用户通常必须输入自己的名字及产品系列号，设置安装目录等等。TaskPage 对象提供了这种界面，每次显示给用户一个 TaskSection，允许用户向前和向后移动，并且在任何时候都允许用户取消操作。在最后一步，用户可以单击“Finish”按钮完成所做的任务。

TaskSection 类只是一个用于两种面板的容器类，包括一个图像面板和一个指示面板，图像面板用来显示一个图像，指示面板描述当前任务。例如，如果有一步需要输入用户地址，则可能将一个标有地址的信封图像放置于图像面板，而指示面板则包括了本步中所有涉及到的字段。

通过传送图形和指示面板到构造器，可以创建一个 TaskSection：

```
public TaskSection(Panel graphics, Panel directions)
```

TaskSection 类是抽象类，所以用户必须创建自己的子类。

TaskPage 类是 TaskSection 对象的容器，用户必须创建它并使用每个新任务节和节的名字调用 addSection 方法：

```
public void addSection(String sectionName, TaskSection section)
```

当用户在任何时候按了任务页下方的按钮之一时，调用 buttonPress 方法：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

public void buttonPress(int whichButton)

whichButton 参数可能的值有:BACK、CANCEL、FINISH、HELP 及 NEXT。

与 TaskSection 类一样,TaskPage 类也是抽象类,所以用户必须创建自己的子类。

程序清单 37.5 给出一个小程序,该小程序概括了 hokey pokey 游戏的步骤并且当按了 Finish 按扭时打印一个对话框。

```
import sunw.admin.avm.base.*;
import java.applet.*;
import java.awt.*;

public class TaskApplet extends Applet
{
    class DirectionsTaskSection extends TaskSection
    {
        DirectionsTaskSection(Panel titlePanel, Panel directionsPanel)
        {
            super(titlePanel, directionsPanel);
        }
    }

    class HokeyPokeyTask extends TaskPage
    {
        public void buttonPress(int whichButton)
        {

// See if the Finish button was the one pressed

            if (whichButton == TaskPage.FINISH) {
                InformationDialog dialog = new InformationDialog(
                    Util.findFrame(this),
                    "That's what it's all about!");
                dialog.setVisible(true);
            }
        }
    }

    protected TaskSection createDirections(String title, String directions)
    {

// Create the title panel
        Panel titlePanel = new Panel();
        titlePanel.setLayout(new FlowLayout());
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        titlePanel.add(new Label(title));

    // Create the directions panel
        Panel directionsPanel = new Panel();
        directionsPanel.setLayout(new GridLayout(1, 0));

        TextArea area = new TextArea(directions, 10, 40);
        area.setEditable(false);

        directionsPanel.add(area);

        return new DirectionsTaskSection(titlePanel, directionsPanel);
    }

    public void init()
    {
        TaskPage page = new HokeyPokeyTask();

        TaskSection section = createDirections(
            "Step 1","Put your left foot in");
        page.addSection("Step 1", section);

        section = createDirections("Step 2", "Take your left foot out");
        page.addSection("Step 2", section);

        section = createDirections("Step 3", "Put your left foot in");
        page.addSection("Step 3", section);

        section = createDirections("Step 4", "Shake it all about");
        page.addSection("Step 4", section);

        section = createDirections("Step 5", "Do the hokey pokey");
        page.addSection("Step 5", section);

        section = createDirections("Step 6", "Turn yourself around");
        page.addSection("Step 6", section);

        setLayout(new GridLayout(1, 0));
        add(page);
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

37.4.5 对话框

对话框是基于表单的用户界面的主要组成部分。AVM 提供了一些常用的对话框，省却了用户自己编程的麻烦。在编写小程序时开发人员所遇到的麻烦事情之一是 AWT 对话框需要一个 Frame 对象。Util 类提供了从一个小程序或其它 AWT 组件获得 Frame 的方便方法。只要在所希望的 Frame 中调用 findFrame，并传送一个组件(如小程序)即可：

```
Frame currentFrame = Util.findFrame(this);
```

InformationDialog InformationDialog 类提供一条文本消息和一个 OK(确认)按钮。它还提供一个图标，表明它是一条信息消息，而不是一条错误消息。创建一个信息对话框有三种不同的方式：

```
public InformationDialog(Frame f)
public InformationDialog(Frame f,String message)
public InformationDialog(Frame f,String message, String title,boolean modal)
```

下述的代码段创建了一个使用“Hello World!”消息的信息框：

```
public InformationDialog dialog = new InformationDialog(
    Util. findFrame(this),"Hello World!");
```

在这个例子中，方式标志为真(true)，表明在显示对话框时，没有活动的父框架(Frame)发生。换句话说，在继续之前用户必须单击 OK(确定)按钮。图 37.2 给出了“Hello World!”对话框。

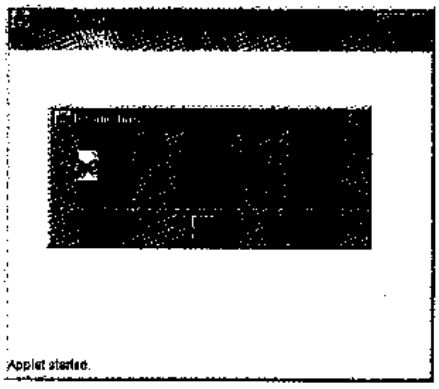


图 37.2 给出一条消息和一个 OK(确认)按钮的信息对话框

ErrorDialog 和 WarningDialog ErrorDialog 类显示一个与 InformationDialog 类几乎完全相同的对话框，只是消息旁边的图标和对话框的标题表明它表示的是一条错误。ErrorDialog 的构造器与 InformationDialog 的构造器具有相同的形式：

```
public ErrorDialog(Frame f)
public ErrorDialog(Frame f,String message)
public ErrorDialog(Frame f,String message,String title,
  boolean modal)
```

图 37.3 给出了一个错误对话框。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

WarningDialog 类与 ErrorDialog 类一样，只是其图标及缺省的标题指出它是一个警告而不是一条错误。WarningDialog 的构造器使用与 ErrorDialog 相同的形式。

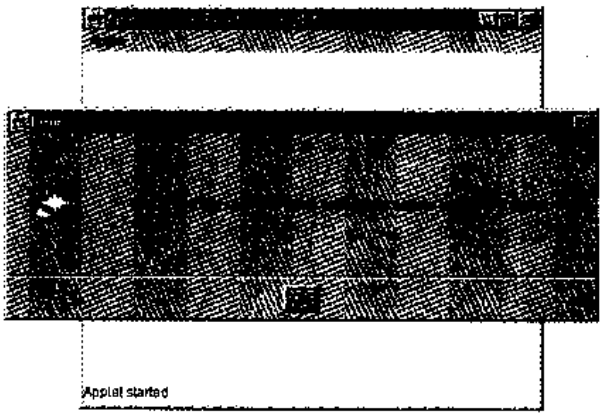


图 37.3　一个错误对话框，它类似于一个信息对话框

QuestionDialog：有时候我们需要对话框比仅仅提供信息做更多的事情。毕竟对话框这一术语指的是双向通信。QuestionDialog 类提供了三个按钮：Yes(是)、No(否)、Cancel(取消)。使用 JavaBeans 的 ActionListener 类可检索来自对话框的结果。程序清单 37.6 给出了一个应用程序，程序提出一个问题，然后在一个信息对话框中显示出结果。

```
import sunw.admin.avm.base.*;
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class QuestionApplet extends Applet implements ActionListener
{
    public void start()
    {
        QuestionDialog dialog = new QuestionDialog(
            Util.findFrame(this), "Do you want to go on the cart?",
            "I'm not dead", true);

        dialog.addActionListener(this);

        dialog.setVisible(true);
    }

    public void actionPerformed(ActionEvent evt)
    {
        Object ob = evt.getSource();

        if (! (ob instanceof QuestionDialog)) return;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        String command = evt.getActionCommand();
        String response = null;

        if (command.equals("yes")) {
            response = "Why yes, I'll go on the cart.";
        } else if (command.equals("no")) {
            response = "I don't want to go on the cart.*;
        } else if (command.equals("cancel")) {
            response = "I'm not dead.";
        }

        InformationDialog info = new InformationDialog(
            Util.findFrame(this), response, "Cart result", true);
        info.setVisible(true);
    }
}
```

图 37.4 给出了活动中的提问小程序。

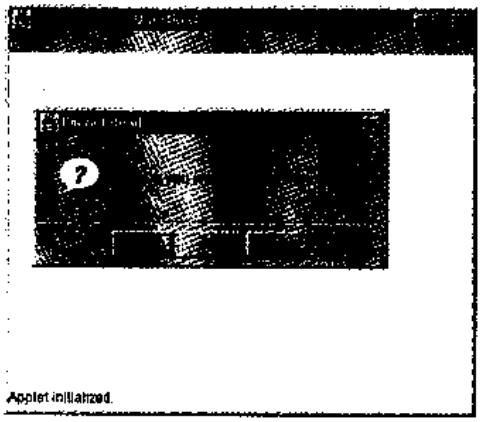


图 37.4 等待 Yes(是)、No(否)或 Cancel(取消)的提问对话框

ButtonDialog QuestionDialog 类实际上只是一般 ButtonDialog 的一个特例。按钮对话框的设计有一边界布局,任何需要显示的信息都应该添加到中央部分。按钮沿着对话框底部显示,可以是如下标准按钮中的任何一个:Apply(应用)、Cancel(取消)、OK(确认)以及 Reset(重置)。缺省情况下,只显示 OK(确认)和 Cancel(取消)按钮。通过对每个按钮调用 SetVisible,我们可以激活和禁止各种按钮。

如果我们创建 ButtonDialog 的一个子类,则可使用如下的局部变量访问各种按钮:

```
protected Button applyButton;
protected Button cancelButton;
protected Button helpButton;
protected Button okButton;
protected Button resetButton;
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

例如，如果我们需要激活 ButtonDialog 子类中的 Help(帮助)按钮，只要以如下方式调用 setVisible 即可：

```
helpButton.setVisible(true);
```

因为我们有时需要使用未创建子类的对话框(并且，如果从 Java 1.0 开始，读者就一直在使用 Java AWT，可能会讨厌创建子类)，可以使用如下的方法定位所需的按钮：

```
public Button getApplyButton()
public Button getCancelButton()
public Button getHelpButton()
public Button getOKButton()
public Button getResetButton()
```

与 QuestionDialog 一样，我们可以通过添加一个动作侦听器和检验动作命令，获得 ButtonDialog 的结果。命令的值将是 apply(应用)、cancel(取消)、help(帮助)、ok(确认)及 reset(重置)——正如我们所希望的。

ProgressDialog　当一个操作可能持续很长时间，或只是想在操作进行时持续提示用户，在这些情况下我们可以使用 ProgressDialog 对象，该对象在屏幕显示一个进展条。一般来说，调用 setValues 可以设置对话框的启动、结束及当前值，然后再调用 setValue 更新当前值。同样，如果调用 setIntervals，可以使得进展条看起来像一系列的小方块，而不是一条实线。程序清单 37.7 给出了一个进展对话框小程序，该小程序使用一个线程更新对话框。

```
import sunw.admin.avm.base.*;
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class GaugeApplet extends Applet implements ActionListener
{
    ProgressDialog progress;
    Thread progressThread;
    boolean done;

    class ProgressThread implements Runnable
    {
        public void run()
        {
            int direction = 1;
            int value = 0;
            done = false;

            while (! done) {
                value = value + direction;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                if (value < 0) {
                    value = 0;
                    direction = 1;
                } else if (value > 100) {
                    value = 100;
                    direction = -1;
                }
                progress.setValue(value);
                try {
                    Thread.sleep(250);
                } catch (Exception ignore) {
                }
            }
        }
    }
    public void init()
    {
        progress = new ProgressDialog(Util.findFrame(this),
            "Making progress");
        progress.setValues(0, 0, 100);
        progress.setIntervals(10);
        progress.addActionListener(this);

    }

    public void start()
    {
        progressThread = new Thread(new ProgressThread());
        progressThread.start();
        progress.setVisible(true);
    }

    public void stop()
    {
        if (progressThread != null) {
            done = true;
            progressThread = null;
            progress.setVisible(false);
        }
    }

    public void actionPerformed(ActionEvent evt)
    {
        Object ob = evt.getSource();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        if (! (ob instanceof ProgressDialog)) return;

        if (progressThread ! = null) {

            done = true;
            progressThread = null;
            progress.setVisible(false);
        }
    }
}
```

图 37.5 给出了活动中的 GaugeApplet 程序。

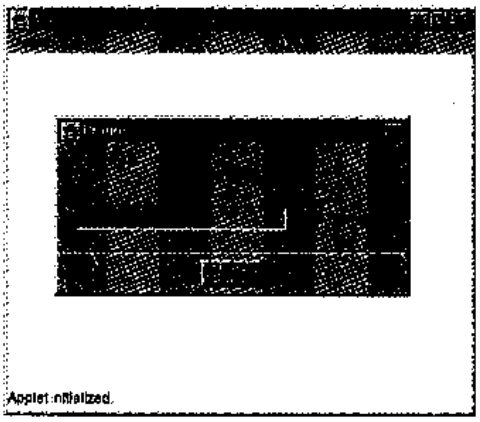


图 37.5　显示一个进展条和一条消息的进展对话框

37.4.6　自验证字段

AWT 的 TextField 类对于获得用户信息非常有用，当所需要的信息不是文本字符串时，就必须自己编码以确保字段中数据采用正确的格式。由于经常需要读取整型数、双精度型数及日期型数据，AVM 提供了 DateField 、IntegerField 及 DoudleField 类，自动保证字段内的文本采用正确的格式。

IntegerField　整型字段只接收数字 0－9 并通过 getValue 方法返回当前值：

```
public int getValue();
```

同样，setValue 方法允许用户修改字段的值：

```
public void setValue(int newValue);
```

创建整型字段有几种方法。我们可以创建一个空字段，可以使用一个整数或字符串初始化字段，也可以在字段中设置列的最大个数。IntegerField 的构造器如下所示：

```
public IntegerField()
public IntegerField(int value)
public IntegerField(String text)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public IntegerField(int value,int cols)
public IntegerField(String text,int cols)
```

DoubleField DoubleField 类几乎与 IntegerField 一样，只是 DoubleField 接收的是浮点数。正如 IntegerField 类一样，getValue 方法返回当前值，setValue 方法则修改字段的值：

```
public double getValue();
public void setValue(double newValue);
```

DoubleField 的构造器与 IntegerField 的构造器形式相同，只是其初始化值指定的是双精度型而不是整型：

```
public DoubleField()
public DoubleField(double value)
public DoubleField(String text)
public DoubleField(double value,int cols)
public DoubleField(String text,int cols)
```

DataField DataField 在文本字段中提供合法的日期值。由于日期比简单数据稍复杂一些，DataField 类比 IntegerField 和 DoubleField 类要复杂。虽然日期型比相应的数值型有更多的方法，但 DataField 构造器实际上没有什么大的变化：

```
public DataField()
public DataField(Date date)
public DataField(String date)
```

我们可以使用 getDate 方法以日期对象的形式检索当前值；也可以使用 getDateString 方法以字符串形式检索当前值：

```
public Date getDate()
public String getDateString()
```

我们也可以以日期对象或字符串方式设置日期：

```
public void setDate(Date newValue)
public void setDate(String newValue)
```

我们还可以单独操作日期的单个组件：

```
public int getDay()
public int getMonth()
public int getYear()

public boolean setDay(int day)
public boolean setDay(String day)
public boolean setMonth(int month)
public boolean setMonth(String month)
public boolean setYear(int year)
public boolean setYear(String year)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

37.4.7　新布局管理器

对于所有新的面板，AVW 有一些确实有用的新布局管理器：BulletinLayout 、RowLayout、ColumnLayout 以及 FieldLayout。有些布局管理器与在网络管理应用程序下所经常看到的用户界面类型相似。

BulletinLayout　在面板中布局组件的一个鲜为人知的方法是使用 null 布局管理器。如果我们将一个面板中的布局管理器设置为 null，则可以使用 setLocation、setSize 及 setBounds 方法手工移动组件。显然，有些使用 AVM 设计的人员认定 null 布局管理器必须从一个储藏室中产生出来，而现在我们有了 BulletinLayout（可将其看作是 Nulletin 布局）。当我们设置其面板的布局管理器为 BulletinLayout 后，只需添加组件并设置其位置即可，如下所示：

```
setLayout(new BulletinLayout());

button b1 = new Button("Push Me");
add(b1);
b1.setSize(100,75);
b1.setLocation(200,200);

button b2 = new Button("No, Push me!");
add(b2);
b2.setBounds(30, 250, 100, 50);
```

如果我们担心 null 布局管理器是否会从将来的 Java 版本中取消，那么使用 BulletinLayout——它得到支持的机会要大一些。

Rowlayout 和 ColumnLayout　许多时候我们只想设计一个所有字段都在一行或一列的布局，我们可以使用网格（grid）布局。但是如果不知道究竟有多少项时怎么办呢？如果添加太多的项，则网络布局将启动一个新的行或列。RowLayout 和 ColumnLayout 类可使我们创建一个具有很多项的长行或长列。行的高度由最高的组件决定，而列的宽度则由最宽的组件确定。

当我们创建一个行或列布局时，可以指定一个布局对齐方法，以指明每个组件将如何对齐。这些值指示组件左对齐、右对齐或居中等。LAYOUT _ ALIGNMENT 方法定义了可能的布局值，包括 CENTER、LEFT、RIGHT、TOP、BOTTOM、FIT 及 EXPAND。

RowLayout 的构造器如下：

```
public RowLayout();
public RowLayout(LAYOUT _ ALIGNMENT alignment);
public RowLayout(LAYOUT _ ALIGNMENT alignment, int hgap, int vgap);
```

同样，ColumnLayout 的构造器为：

```
public ColumnLayout();
public ColumnLayout(LAYOUT _ ALIGNMENT alignment);
public ColumnLayout(LAYOUT _ ALIGNMENT alignment, int hgap, int vgap);
```

当我们设置完这些布局管理器后，可以使用一般的 add 方法，不使用任何特殊参数来添加组件。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

ButtonLayout ButtonLayout 类通常用于设置一个按钮行，但是它能够处理任何类型的组件。ButtonLayout 在安排组件时自左向右在一行上安排，并在每行上居中。当一行上安装不下更多的组件时，将移动到下一行。与 RowLayout 和 ColumnLayout 类一样，ButtonLayout 的构造器也有三种格式：

```
public ButtonLayout();
public ButtonLayout(LAYOUT_ALIGNMENT alignment);
public ButtonLayout(LAYOUT_ALIGNMENT alignment, int hgap, int vgap);
```

FieldLayout FieldLayout 类非常简单但却提供了非常有用的功能。当我们创建数据录入表单时，一般来说是创建一个标签和一个输入字段。FieldLayout 类假设我们需要的正是这样的设计，等待着我们一一添加标签以及字段。下述的代码段添加了一系列的字段及其标签：

```
setLayout(new FieldLayout());
add(new Label("First Name: "));
textField firstNameField = new TextField(20);
add(firstNameField);
add(new Label("Last Name: "));
textField lastNameField = new TextField(20);
add(lastNameField);
```

FieldLayout 对象没有对齐选项，但是如果愿意，我们仍可以在构造器中指定 hgap(水平间隔)和 vgap(垂直间隔)值：

```
public FieldLayout();
public FieldLayout(int hgap,int vgap);
```

自从设计 Admin View Module 以来，其中的一部分已经包括在了 Java 基类(Java Fundation Classes，JFC)中，其余的部分也可能会在将来的某一天也被包括到 JFC 中，但这仍有待于观望。目前，AVM 对于在 JMAPI 框架内显示和操作管理对象提供了一个有用的框架。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第五部分　数　据　库

第 38 章　数据库介绍
第 39 章　JDBC:Java 数据库连接
第 40 章　JDBC 考察

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 38 章　数据库介绍

本章内容：

38.1　ODBC 和 JDBC
38.2　关系数据库概念
38.3　ODBC 技术概述
38.4　高级客户/服务器概念

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

38.1 ODBC 和 JDBC

本章是关于处理 Java 程序对数据库访问的三部曲中的第一章:数据库介绍。标准的关系数据访问对 Java 程序是很重要的,因为 Java 小程序的本质并不是单一集成的、全能的应用程序。由于 Java 小程序本质上是用标准化组件组装的,它们需要从数据存储器中读取固定数据,对数据进行处理,并把数据写回到数据存储器,以便于其它 Java 小程序进行处理。单一集成的程序可以提供其自己特有的数据处理模式,但是由于 Java 小程序是跨操作系统和跨网络平台的,所以需要使用公布的开放数据访问模式。

Java 数据库连接(Java Database Connectivity,JDBC)是 Java 企业版 API 的一部分,提供了从 Java 程序跨平台、跨数据库的数据库访问。企业版 API 还包括远程调用(Remote Method Invocation,RMI)和系列化 API(Java 程序跨名字空间的对象管理和远程对象中的方法调用),用于与 CORBA 和其它面向对象的系统通信的 Java IDL(接口定义语言,Interface Definition Language),以及用于访问跨企业命名和目录服务的 Java JNDI(Java 命名和目录接口,Java Naming and Directory Interface)。

本章将介绍关系的概念,以及微软公司的开放式数据库连接(ODBC)。本章介绍 ODBC 主要有两个方面的原因:

- JDBC 和 ODBC 均基于 SAG CLI(SQL Access Group Call Level Interface)规范。
- JOBC 设计主要使用了来自 ODBC 的抽象和方法。

把 JDBC 的设计基于 ODBC 的原因是:ODBC 在 ISV(Independent Software Vendor,独立软件开发商)和用户中非常流行,对于以前具有 ODBC 经验的数据库开发人员来说,实现和使用 JDBC 将会更容易。另外,Sun 公司和 Intersolv 公司已经开发出了一个 JDBC – ODBC 桥接层,以便更好地利用市场上可用的 ODBC 驱动程序。所以使用 JDBC API 和 JDBC – ODBC 桥接器,我们可以从 Java 小程序和应用程序,对几乎所有的数据库进行更有效的访问和交互操作。

38.2 关系数据库概念

正如我们所知道的那样,数据库包含以特殊方式组织的数据。数据库可以是一个非常简单的展开文件(含有以制表形式存放的单一计算机文件),包括朋友的名字及电话号码;也可以是一个非常精细的世界范围的主要航班定票系统。本章所讨论的许多原理对于绝大多数的数据库系统都是适用的。

从结构上看,数据库主要有三种类型:

- 层次型数据库
- 关系型数据库
- 网状数据库

在 70 年代和 80 年代期间,层次型结构非常流行。这种结构把数据看成一个树状结构的系统,数据记录形成树的叶。层次型结构的实现实例是二叉树结构。层次型结构中最常见的关系是数据记录间一对多关系,如果没有数据冗余,它很难实现多对多的关系。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

数据库的关系类型

在数据库表中建立和保持数据记录之间的关系要比维持人之间的关系困难得多！

在记录之间有三种类型的数据记录关系：

- 一对一关系：一个表中的一条记录与另一个表中的至多一条记录有关系。反之亦然。一本书与其 ISBN 之间的关系（一本书只有一个 ISBN，并且一个 ISBN 只与一本书相关联）是一个一对一关系很好的实例。
- 一对多关系：一个表中的一条记录可与另一表中的多条记录相关联。购货订单与各单项之间的关系（一个购货订单可以有多个单项，但一个单项只能与一个购货订单相关联）是一个一对多关系的实例。
- 多对多关系：这和学生与课程之间的关系类似（一个学生在一个学期中可以学习不同老师的多门课程，并且一门课程有多个学生学习）。

我们可能不知道数据库是如何记住这些数据关系的。这通常是使用在两个表中同时存放一个公共元素如学生 ID 号/课程 ID 号，或使用保存两个记录的一张记录 ID 表（称为索引）的方法来实现的。现代数据库有很多其它的复杂手段保持数据记录关系的完整性，不论是对更新、还是删除等操作。

网状数据模型是通过假设数据元素之间存在一个多重关系来解决这一问题的。与层次模型相反，层次模型中存在的关系是父子关系，而在网状模型中存在的关系是点对点关系。当时所开发的大部分程序采用了层次和网状相结合的数据存储和访问模式。

90 年代期间，关系型数据访问模式开始成为主流。关系模式则以信息行的方式看待数据，每一行包含许多数据列，数据列称为字段。关系型模式中最主要的概念是数据具有统一的形式，每一行都包含相同个数的列。一个这样的行和列的集合称为一个表。许多这样的表（其结构可能不同）形成一个关系型数据库。

图 38.1 给出了一个关系型数据库模式（或称表设计）的样例，它是一个关于学生登记的数据库。在该实例中，数据库由三个表组成：学生（Student）表，它包含了有关学生的信息；课程（Course）表，它包含了有关课程的信息；以及学生课程（StudentCourse）表：它给出了学生与课程之间的关系。学生表包含的信息有学生号、姓名、地址等；课程表包含的信息包括课程号、课程名称、开课位置等。

现在有了带有数据的学生表和课程表，如何将这些表关联起来呢？这就是关系型数据库中关系部分所要实现的部分。为了使两个表关联起来，两个表都必须含有一个公共列，或者必须创建第三个表，其中包含两个列，一个列来自于第一个表，另一个列来自于第二个表。

让我们来看看这是如何实现的。在该实例中，为了使学生表与课程表关联起来，我们需要创建一个新表，即学生课程（StudentCourse）表，其中包含两个列：学生号和课程号。当一个学生选修了一门课程后，学生课程表中就使用该学生的学生号和该课程的课程号产生了一个新行。这样该表就有了学生与课程之间的关系。如果我们需要查找一组学生及其所选修的课程，进入学生课程表，读取每一行，在学生表中查找与学生姓名对应的学生号，在课程表中查找与课程名称对应的课程号，最后选出学生姓名和课程名称两列即可。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

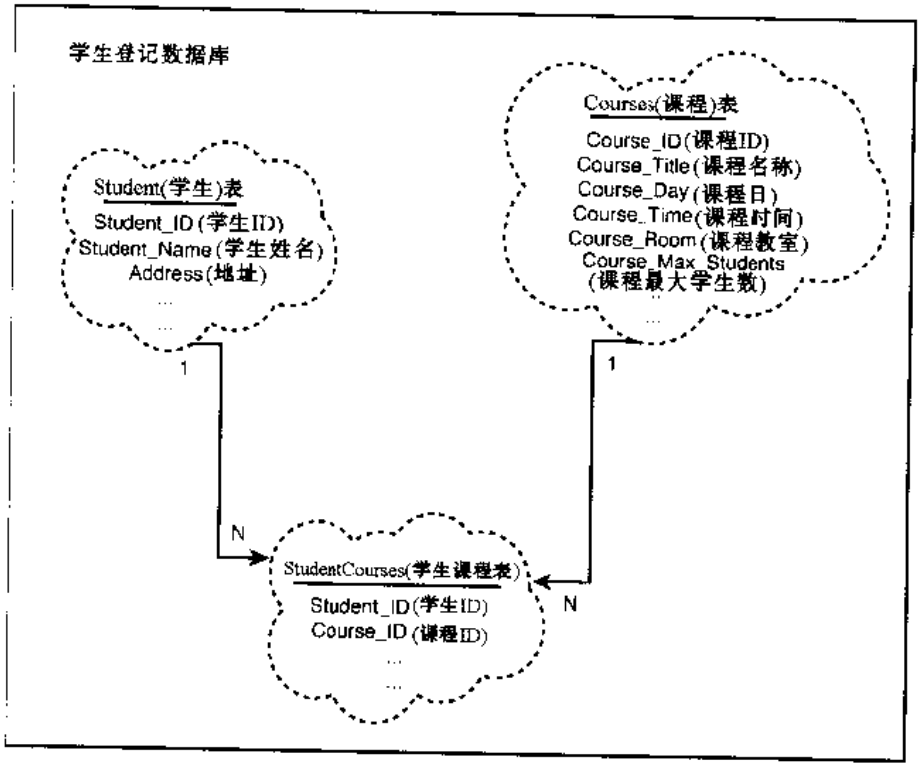


图 38.1 关系型数据库模式实例:一个学生登记数据库

38.2.1 SQL

在关系型数据库变得越来越流行的情况下,数据库专家们需要一种通用的数据库语言来执行数据操作。结果出现了结构化查询语言(Structured Query Language,SQL)。SQL 已经发展成了一种主流数据库语言,它包括了数据操作功能如创建、更新及删除,数据定义功能如创建表和列,安全性功能如限制访问数据元素、创建用户和组,数据管理功能如数据备份、批量复制及批量更新,以及最为重要的事务处理功能。SQL 与编程语言 Java、C++及其它语言一起使用,并用于数据处理及与后台数据管理系统进行交互。

注意:每个数据库厂商都有其自己的 SQL 实现。微软的 SQL Server 是一种客户/服务器关系型 DBMS,它的 SQL 称为 Transact-SQL(事务处理 SQL),而 Oracle 的 SQL 称为 PL/SQL。

1986 年,SQL 成为了 ANSI(American National Standards Institute)标准,后来修订为 SQL-92。JDBC 遵守的是 SQL-92。

38.2.2 连接

数据库是由具有数据行的表组成,这并不意味着我们只能在数据库固定的表内观察数据。连接是指将两个或多个表结合在一起形成一个单一表的过程。连接可以是动态的,这种情况下两个表被拼结成一个虚拟表;连接也可以是静态的,这种情况下两个表连接并保存起来供以后使用。静态连接通常是一个存储的过程,可以通过激活该过程来刷新所保存的表,然后查询所保存的表。连接是在具有公共信息列的表上操作的。从理论上讲,连接有许多种,在本节的后面我们将进行讨论。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

在深入了解连接之前，让我们先来看看下面的例子，在这个例子中我们将使用表 38.1、表 38.2 和表 38.3 中所给出的新的记录填写图 38.1 中数据库模型中的表。这些表只给出了有关的字段或列。

表 38.1　学生表

Student _ ID(学生 ID)	Student _ name (学生姓名)
1	John
2	Mary
3	Jan
4	Jack

表 38.2　课程表

Course _ ID(课程号)	Course _ Title(课程名称)
S1	Math
S2	English
S3	Computer
S4	Logic

表 38.3　学生课程表

Student _ ID(学生号)	Course _ ID(课程号)
2	S2
3	S1
4	S3

内连接(Inner Join)　内连接是一种简单连接，学生表和学生课程表之间的一个内连接得到一个如表 38.4 所示的表。我们通过将 Student _ Name 列添加到学生课程表的方法，将学生表和学生课程表结合起来得到一个新表。

表 38.4　内连接表

Student _ ID(学生 ID)	Student _ Name(学生姓名)	Course _ ID(课程 ID)
2	Mary	S2
3	Jan	S1
4	Jack	S3

使用 Student _ ID 连接两个表并不意味着我们应提取这一列的数据。我们可以将关键字从一个内连接的结果中排除出去。这一内连接的 SQL 语句如下所示：

```
SELECT Students.Student _ Name, StudentsCourses.Course _ ID
From Students, StudentsCourses
WHERE Students.Student _ ID = StudentsCourses.Student _ ID
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

外连接(Outer Join) 对于两个表(如 Table1 和 Table2),当连接结果表中包含第一个表中的所有行和第二表中的公共行时,这种连接为外连接。(所谓第一个表和第二个表是由其在 SQL 语句中的顺序决定的)。例如假设一个 SQL 语句中有 FROM Table1, Table2 子句,则一个左外连接将选择出第一个表(Table1)中的所有行和第二个表(Table2)中的公共行。在一个右外连接中将选择出第二个表(Table2)中的所有行和第一个表(Table1)中的公共行。使用学生表和学生课程表进行左外连接将产生一个如表 38.5 所示的表。

表 38.5 外连接表

Student _ ID(学生 ID)	Student _ Name(学生姓名)	Course _ ID(课程 ID)
1	John	< null >
2	Mary	S2
3	Jan	S1
4	Jack	S3

当我们需要所有学生的姓名而不管其本学期是否选修了课程,以及本学期所注册的学生选修课程时,这种连接很有用。有人称这种连接为 if _ any 连接,就像是"给我一份所有学生的名单和他们正在选修的课程,如果学生选修了的话"。

上述这一外连接的 SQL 语句如下所示:

```
SELECT Students.Student _ ID, Students.Student  Name, StudentsCourses.Course _ ID
FROM (
oj c: \ enrol.mdb Students
LEFT OUTER JOIN c: \ enrol.mdb
StudentsCourses ON Students.Student _ ID = StudentsCourses.Student _ ID
```

正像我们所猜测到的,完全外连接返回拼接公共行的两个表中的所有记录,如表 38.6 所示。

表 38.6 完全外连接表

Student _ ID(学生 ID)	Student _ Name(学生姓名)	Course _ ID(课程 ID)
1	John	< null >
2	Mary	S2
3	Jan	S1
4	Jack	S3
< null >	< null >	S4

减法连接(Substract Join) 如果我们只想要这学期没有登记的学生或没有学生选修的课程(不受欢迎的课程或教师),怎么办?我们可以使用减法连接。在这种情况下,连接返回不在另一个表中的行。切记,减法连接只有来自于第一个表中的字段。SQL 语句如下所示:

```
SELECT Students.Student _ Name
FROM (
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
oj c: \ enrol.mdb Students
LEFT OUTER JOIN c: \ enrol.mdb
StudentsCourses ON Students.Student _ ID = StudentsCourses.Student _ ID
WHERE (StudentsCourses.Course _ ID Is Null)
```

连接和 SQL 语句　还有许多其它类型的连接，如自连接是一个具有相同的结构的两个表的左外连接。一个自连接的例子是制造业中在一个材料应用帐单中组装/组件的剧增。但是，通常情况下对于一般的应用我们已经学习过的连接类型已经足够了。随着使用 SQL 语句经验的增加，将会开发出更加特殊的连接。

在所有这些连接中，我们都对含有相同的值的列进行比较，这些连接称为等值连接(equi - join)。实际上连接并不只限于列的等值比较。我们可以基于列值的条件(例如--个表的列值大于另一个表的列值)进行两个表的连接。

对于等值连接，由于列值是相等的，我们只检索出公共列的一份拷贝，所以这样的连接有时也被称为自然连接(natural join)。在使用非等值连接时，我们可能需要从两个表中检索公共列。

当一个 SQL 语句到达一个数据库管理系统时，DBMS 将分析 SQL 语句语法，并将 SQL 语名翻译成一个称为查询计划(guery plan)的内部模式，从数据库的表中检索数据。在所有客户/服务器数据库中，这种内部模式生成器都包含一个优化模块，这种优化模块了解其数据库实现的限制和优点。

在许多数据库如微软的 SQL Server 中，优化器是一个基于消耗的查询优化器。当给定一个查询时，该优化器生成多个查询计划，计算每个计划的消耗估计(知道数据存储模式，页面 I/O 等)，然后确定出检索数据效率最高的访问方法，包括表的连接顺序和索引的用法。这种优化后的查询被转换为一种二进制的形式，称之为执行计划(excution plan)，该执行计划施行于数据，以获得结果。常常有这样的情况，即有些未经优化的查询直接执行时需要花费几个小时，而经过优化器实现优化后再执行只需要几分钟就可以了。所有主要的客户/ 服务器数据库中都有内置的查询优化模块，用于处理所有的查询。数据库系统管理员可以指定参数值如消耗、存储模式等，调整其优化器的性能。

38.3　ODBC 技术概述

ODBC 技术是 PC 领域内最流行的数据库接口之一，并且正在逐渐地扩展到其它平台。ODBC 是微软公司对 X/Open 和 SQL 访问组调用层接口(SQL Access Group Call Level Interface)规范的实现。ODBC 提供了编程语言与数据库进行交互的功能，包括添加、修改及删除数据，获得关于数据库表、视图及索引的详细内容等。

> **提示**：此外对于 ODBC 的讨论是与 Java 及 JDBC 的观点有关的。注意到 JDBC 和 ODBC 体系结构之间的相同点和不同点将是很有帮助的。另外，通过 ODBC 的学习可能会给我们一些启示，如 JDBC 未来的发展方向。

图 38.2 给出了 ODBC 体系结构的示意图。ODBC 应用程序有五个逻辑层：应用层、ODBC

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接口、驱动程序管理器、驱动程序以及数据源。

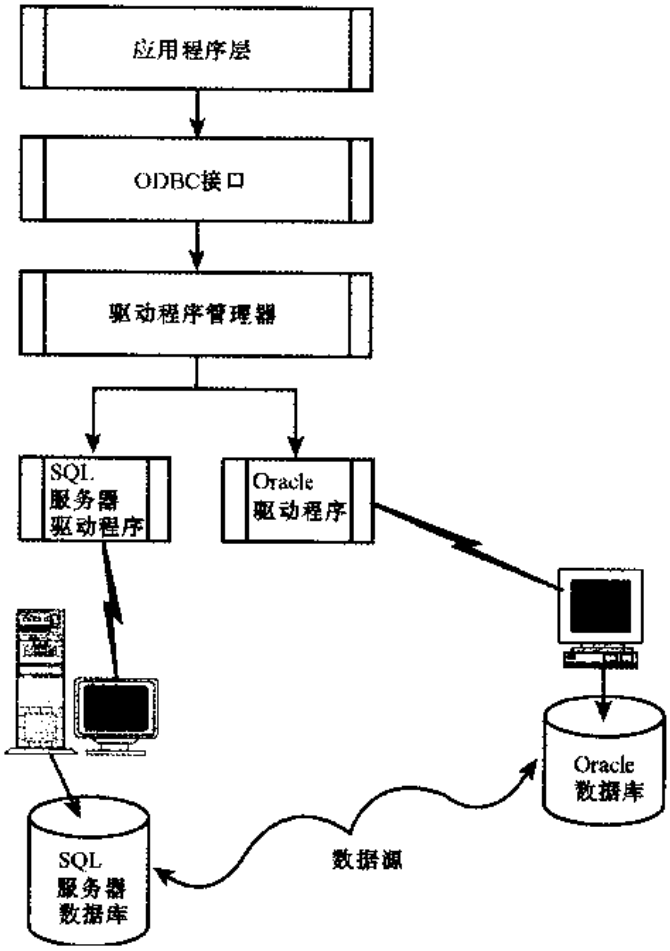


图 38.2 五个 ODBC 层的结构示意图

应用层提供 GUI 和事物逻辑，它是使用诸如 Java、Visual Basic 及 C++ 这样的语言编写的。应用层使用 ODBC 接口中的 ODBC 功能与数据库进行交互。

驱动程序管理层是微软公司 OBDC 的一部分。正如其名字所表示的，驱动程序管理器系统中现存的各种驱动程序，包括加载和定向调用正确的驱动程序，当应用层需要时提供驱动程序信息等。一个应用程序可以连接到多个数据库（例如遗产系统和部门数据库），驱动程序管理器确保正确的 DBMS 能够得到指向它的所有程序的调用，并保证来自数据源的数据被传送给应用程序。

驱动程序是真正懂得特定数据库的组件。通常一个驱动程序归属于一个特殊的数据库，如 Access 驱动程序、SQL Server 驱动程序或 Oracle 驱动程序等。ODBC 接口有一套诸如 SQL 语句、连接管理、关于数据库的信息等调用。实现所有这些功能都是驱动程序的任务。这对于有些数据库意味着，它的驱动程序必须模拟实现 DBMS 底层不支持的 ODBC 接口功能。驱动程序负责向数据库发送查询、取回数据以及发送数据到应用程序等。对于局域 Internet 上的数据库，驱动程序还负责处理网络通信。

在 ODBC 中，数据源可以是一个数据库管理系统，也可以只是一般的存储数据，这种情况通常是硬盘上的一组文件。数据源可以是关于某一小型公司开支数据的一个非常简单的 Microsoft Access 数据库，也可以是一个电话公司包括多个服务器，由多达数十亿字节的所有客户详细帐单组成的复杂的数据存储。数据源可能是处理一个数据仓库，也可能是处理一个简单

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的客户清单。

38.3.1　ODBC 一致层

ODBC 的主要部分是驱动程序，它了解 DBMS 并与数据库进行通信。ODBC 并不要求驱动程序支持 ODBC 接口中的所有功能。相反，ODBC 为驱动程序定义了 API 和 SQL 语法的一致层。ODBC 对驱动程序的唯一要求是，当驱动程序符合某一层时，它应该支持在该层上 ODBC 所定义的所有功能，而不管底层数据库是否支持这些功能。

> **注意**：ODBC 驱动程序规范没有设定所支持功能的上限。这指的是符合第一层的驱动程序可以支持一些第二层的功能。而驱动程序仍被看作是第一层一致的，因为它并没有支持所有第二层的功能，而应用程序则可以使用该驱动程序所提供的那些支持第二层的功能。

正如在 ODBC 技术概述中所提到的那样，模拟底层 DBMS 不支持的 ODBC 功能，以便使 ODBC 接口与 DBMS 的实现相隔离，这正是驱动程序的任务。就所关心的 ODBC 接口和应用程序来说，与 ODBC 层的一致意味着所有的功能可以使用，而不管底层 DBMS 的具体情况如何。

> **注意**：应用程序使用 API 调用如 SQLGetFunctions 和 SQLGetInfo，来获得驱动程序所支持的功能。

表 38.7 总结了 API 和 SQL 的各一致层。

表 38.7　API 和 SQL 的 ODBC 一致层

类型	一致层	描述
API 一致层	内核(Core)	所有 SAG CLI 规范中的功能。分配和释放连接、语句及环境处理。准备和执行 SQL 语句。检索结果集和关于结果集的信息。检索错误信息。提交和回退事务功能
	第一层(Level1)	扩展集 1(Extended Set 1)是内核 API(Core API)加上发送和检索部分数据集、检索分类信息、获得驱动程序和数据库功能等等
	第二层(Level2)	扩展集 2 (Extended Set 2)是第一层(Level 1)加上处理作为参数的数组、滚动光标、调用事务 DLL 等功能
SQL 语法一致层	最小语法 (Minimum Grammar)	数据定义语言(Data Definition Language)中的 CREATE TABLE 和 DROP TABLE 功能。数据操纵语言(Data Manipulation Language) 中的 SELECT、INSERT、UPDATE 及 DELETE 功能(简单部分) 简单表达式
	内核语法 (Core Grammar)	与 SAG CAE 1992 规范一致。最小语法(Minimum Grammar)加上 DDL 中的 ALTER TABLE、CREATE INDEX、DROP INDEX、CREATE VIEW 及 DROP VIEW DML 中完整的 SELECT 语句功能。表达式中像 SUM 和 MAX 这样的函数
	扩展语法 (Extended Grammar)	在内核语法(Core Grammar)中增加了诸如外连接、定位 UPDATE、DELETE、更多的表达式、更多的数据类型、过程调用等功能

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

38.3.2 ODBC 函数和命令集

ODBC 有一个丰富的函数集,其范围从简单的连接语句到处理多结果集的存储过程。所有的 ODBC 函数都有一个 SQL 前缀,并且可以有一个或多个参数,这些参数可以具有输入(到驱动程序)或输出(从驱动程序)的类型。让我们来看看连接到一个 ODBC 源需要的一般步骤和实际 ODBC 命令顺序。

典型的 ODBC 步骤 在一个典型的 ODBC 程序中,第一步是使用 SQLAllocEnv(< envHandle >)和 SQLAllocConnect(< envHandle > , < databaseHandle >)函数,分配环境和连接句柄。当获得有效的数据库句柄后,可使用 SQLSetConnectOption(< databaseHandle > , < optionName > , < optionValue >) 设置各种选项。最后可使用 SQLConnect (< dataSourceName > , < UID > , < PW > , ... 等)与数据库连接。

ODBC 命令顺序 现在我们可以使用语句进行工作了。图 38.3 给出了 ODBC 命令顺序:连接到数据库、执行 SQL 语句、处理结果数据及关闭连接。

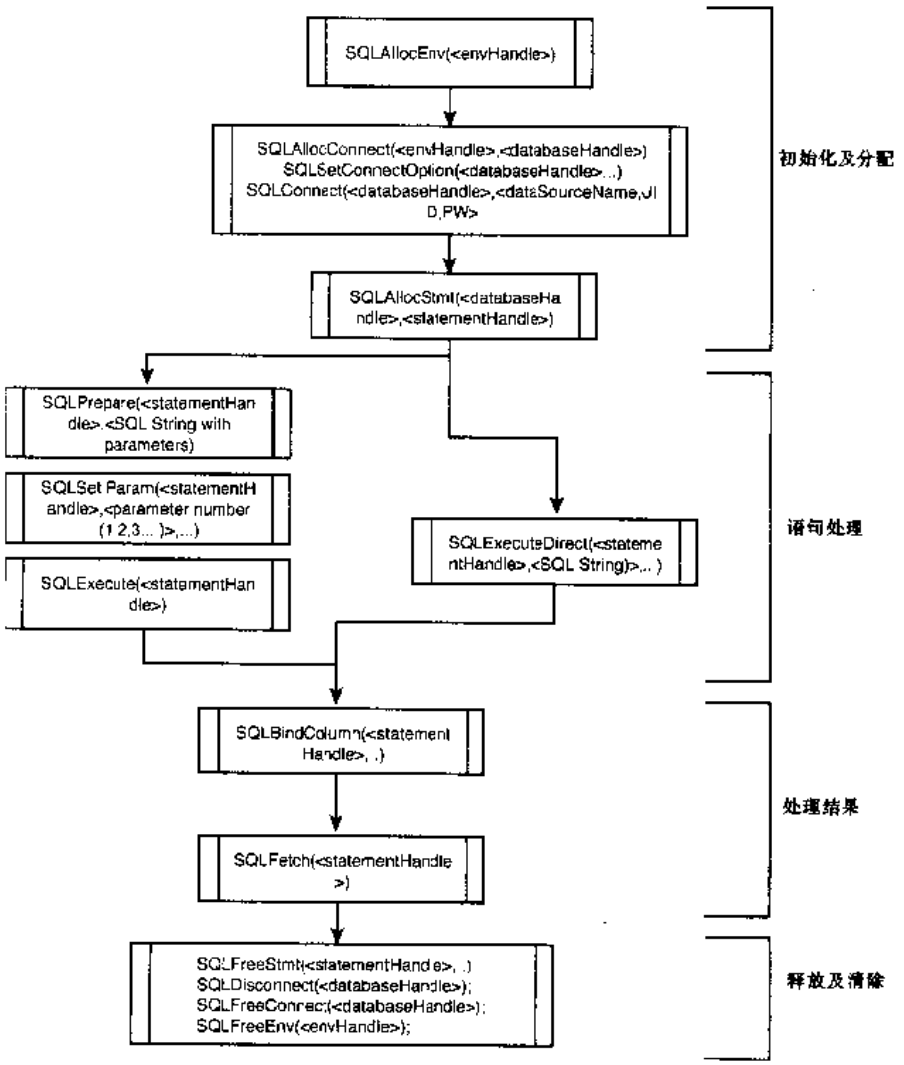


图 38.3 一个典型程序的 ODBC 程序流程示意图

首先,使用 SQLAllocStmt(< datadaseHandle > , < statementHandle >)分配语句句柄,当语句句柄分配之后,可以使用 SQLExecDirect 函数执行 SQL 语句,或使用 SQLPrepare 准备一个

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

语句，然后使用 SQLExecDirect 函数直接执行。我们首先将各列绑定到程序变量，然后通过使用一个 SQLFetch 语句之后，读取这些变量获取一个数据行。当已经没有数据时，SQLFetch 返回 SQL_NO_DATA_FOUND。

> **注意**：对于处理语句和数据，JDBC 也遵守相似的策略。但 JDBC 在数据绑定到变量时是不同的。

现在，我们已经处理完数据，应该释放句柄并关闭数据库了，所使用的语句序列如下：

```
SQLFreeStmt( < statementHandle > ,..);
SQLDisconnect( < databaseHandle > );
SQLFreeConnect( < databaseHandle > );
SQLFreeEnv( < envHandle > );
```

> **注意**：在 JDBC 中，不需要分配语句和句柄。
>
> 这是因为 Java 是一种面向对象的语言，我们获得连接对象后，该连接对象将会给出语句对象。
>
> 由于 Java 具有自动垃圾收集功能，我们无需释放句柄、删除对象等。当一个对象失去其作用域后，JVM 虚拟机将作为自动垃圾收集的一部分，收回该对象所使用的内存。

38.4　高级客户/服务器概念

一个典型的客户/服务器系统至少是一个部门级的系统，更可能是一个跨越一个组织内部许多部门的组织级系统。像经纪业务系统、银行系统、制造业系统以及预定系统这样的面向任务(mission - critical)和全线事务(line - of - bussiness)系统均属于这一类。大部分系统是对一个组织内部的，它也跨越客户和供应商。几乎所有的这种系统都运行在一个局域网(LAN)上，它们也具有广域网(WAN)连接和拨入功能。随着 Internet/intranet 和 Java 的出现，这些系统变得越来越复杂了，能够以许多新的方式处理业务。

让我们看看联邦快递(Federal Express)的情况。它的 Web 站点如今能够规划邮包的投递、跟踪一个邮包从投递到交付的过程、并获得交付信息和时间。我们现在正处在一个在线商业活动即将与商业街购物一样平常这样一个时代的起点上。让我们来看看驱动这样的系统运转的一些概念。

38.4.1　客户/服务器系统层

大多数的应用系统都包括具有前端 GUI、事务规则处理及通过 DBMS 的数据访问等功能的模块。事实上，主要的系统如在线预定、银行和经纪、帐单处理等，都包括上千条业务规则、遍布全球的异类数据库、以及上百个的 GUI 系统。这些系统的开发、管理、维护以及改进都要处理上百万条的代码、涉及众多部门、协调全球成百上千个个人用户的工作。多层系统设计和开发的概念可应用于从部门级系统到全球信息系统这样一个广大的范围。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意:在双层和三层的系统中,一个应用程序逻辑上被分为三个部分:

- GUI(图形用户接口)——由屏幕、窗口、按钮、列表框等组成。
- 事务逻辑——程序中处理各种数据元素交互的部分。其所有的处理都是基于数据元素的值进行的,如根据年收入确定信用卡的限额,根据税收表计算收入所得税,或者根据某一类材料的使用情况进行重订购点的计算等。
- DBMS——处理实际的数据存储和检索。

双层系统 作为基本的层,双层系统包括 GUI 和直接访问数据库的事务逻辑。GUI 在客户系统上,而数据库可以在客户系统上或服务器上。通常,GUI 是使用 C++、Visual Basic、Power Builder、Access Basic 及 Lotus Script 这样的语言编写的。数据库系统一般是 Microsoft Access、Lotus Approach、Sybase SQL Anywhere 、Watcom DB Engine 和 Personal Oracle 等。

三层系统 目前多数组织级和部门级的客户/服务器应用程序使用了三层策略,即 GUI、事务逻辑及 DBMS 三个逻辑层次。此处的 GUI 开发工具包括 Visual Basic、C++ 及 Power Builder。中间层的开发工具也倾向于使用 C++ 或 Visal Basic,而后端的数据库则有 Oracle、Microsoft SQL Server 或 Sybase SQL Server。三层概念带来了一个数据库服务器、应用程序服务器及 GUI 客户机这样的新时代。UNIX 、WINDOWS NT 和 Solaris 等操作系统占领着应用程序服务器和数据库服务器的这一领域。而像 Windows 这样的客户机操作系统则是流行的 GUI 前端。

多层系统 现在随着 Internet 和 Java 的使用,一个"网络就是计算机"和"轻客户"这样一种模式转换的时代已经到来。Java 小程序及其对象和方法创建了多层客户/服务器系统这一思想。从理论上讲,Java 小程序可以是一种事务规则、GUI 或 DBMS 接口。每一个 Java 小程序可以认为是一个层。事实上,并不是 Internet 和 Java 第一次引入面向对象和多层次系统的概念。OMG 的 CORBA 和微软的 OLE (现在称为 ActiveX)体系结构均是面向对象模型和多层次系统的倡导者。有了 Java 和 Internet,这些概念的实现更为容易了。

简短来说,系统设计和实现经历了从双层结构到三层结构,再到目前网间互连的由 Java 小程序驱动的多层结构,这样的一个发展过程。

38.4.2 事务

事务这一概念是任何客户/服务器数据库必不可少的部分。一个事务是一组 SQL 语句,用来更新、添加或删除数据库中的行和字段。事务有一个"全攻全守"的特性——如果所有语句都成功执行则提交事务,或者如果有一个语句没有执行成功则回退整个事务。事务处理保证了数据库中数据的完整性和一致性。

注意:JDBC 使用 commit()和 rollback()方法支持事务处理。另外,JDBC 有一个 autocommit()方法,当该方法打开时,自动提交所有修改,当该方法关闭时,Java 程序必须使用 commit()或 rollback()方法使数据的改变生效。

事务 ACID 属性 事务特性使用元操作性(Atomicity)、一致性(Consistency)、独立性(Isolation)和可持续性(Durability)(ACID)等属性术语来描述。

一个事务是元操作的是指它是一个实体。一个事务的所有部分要么全部发生,要么全部

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

不发生，不存在部分事务之说。如果只有部分事务发生，则该事务取消。元操作性是由 commit()或 rollback()方法完成。

一个事务是一致的是因为它不执行任何违背事务逻辑或数据元素之间关系的动作。事务的一致性特性非常重要，特别是当开发一个客户/服务器系统时，因为在这种情况下对于数据存储有很多来自不同系统和对象的事务。如果一个事务破坏了数据的一致性，所有其它事务也将潜在出错，结果是造成全系统范围的崩溃和数据毁坏。

一个事务是独立的是因为事务的结果是自包含的。事务不依赖于任何先导或后续的事务，这是与称为系列化特性的属性相关的，这意味着事务序列是互相独立的，也就是说，一个事务不假定于任何外部序列。

最后，一个事务是可持续的是指事务的效果是永久性的，即使当系统出错时。这意味着永久性存储的某些形式应该是一个事务的一部分。

分布式事务协调器　在事务处理中一个相关的话题是，对于跨越不同种类数据源、系统和对象的协调。当事务在一个关系型数据库中执行完后，我们可以使用 commit()、rollback()、beginTransaction()及 endTransaction()语句来协调整个过程。但是如果一个事务中有多个不同类的系统参与时又怎么办呢？如何处理这样的一个系统呢？作为一个例子，让我们来看看作为 Microsoft SQL Server 6.5 数据库系统一部分的分布式事务协调器(Distributed Transaction Coordinator，DTC)。

在微软的 DTC 中，由一个事务管理器来完成协调工作。资源管理器作为客户它实现了事务对资源的保护——例如，关系数据库和 ODBC 数据源。

应用程序使用事务管理器开始一个事务，然后，使用资源管理器启动事务，使用事务管理器登记步骤(应征)。

事务管理器跟踪所有的应征事务，在多数据源事务处理的最后一步，应用程序调用事务管理器提交或放弃事务。

当应用程序向事务管理器发出一个提交命令时，DTC 执行一个两阶段提交协议：

1. 它查询每一个资源管理器，是否准备好提交。

2. 如果所有资源都已准备好提交，DTC 向所有这些资源发送一条提交信息。

微软的 DTC 是来自于数据库厂商的功能强大的下一代事务协调器的一个例子。随着越来越多的多平台、面向对象的 Java 系统被开发出来，这种事务协调器将变得十分重要。已经有许多中间商在开发面向 Java 的事务系统。

38.4.3　数据光标

一个关系型数据库查询一般返回多行数据，但是一个应用程序一次只处理一行数据。即使一个应用程序能处理多行数据——例如通过将数据显示成表或数据表形式——它也只能处理有限行的数据。同样，数据的更新、修改、删除及添加也是在逐行的基础上处理的。

这就是引出数据光标这一概念的原因。在此处的上下文中，一个数据光标是指向一个行的指针。这很像屏幕(CRT)上的光标——即一个位置批示器。

数据并发和数据光标模式

根据数据并发性，不同类型的多用户应用程序需要不同类型的数据集。有些应用程序需要立即知道底层数据库的改动情况。例如，在一个预定票系统中，坐位分配信息的动态特性是

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

非常重要的。其它的应用程序，例如统计报表系统，需要的则是稳定的数据，如果数据不断地变化，那么这些程序就无法有效地显示出任何结果。为了支持各种类型的应用程序的需求设计了不同类型的数据光标。

数据光标可以看作是底层的数据缓冲区。一个完全的可滚动式的数据光标就是程序可以在数据缓冲区中可以向前和向后在行上移动的数据光标。如果程序能够更新数据光标中的数据，则这个数据光标称为可滚动的和可更新的光标。

警告：在考虑数据光标时，需要记住的很重要的一点是事务的独立性。当一个用户正在更新一个行时，另一用户可能正在其自己的数据光标内观察该行数据。数据的一致性在这里很重要。糟糕的情况是，第二个用户也可能正在修改同一行数据！

注意：JDBC API 中的 ResultSet 是一个数据光标，但它只是一个可向前流动的数据光标——这就是说用户只能使用 getNext()方法向前移动。

ODBC数据光标类型 ODBC 的数据光标在更新能力、并发性、数据完整性以及功能上都是非常强大的。ODBC 数据光标允许定位删除和定位更新，还可以多行提取(称为一个行集合)，防止更新丢失。

ODBC 支持静态的、键值驱动的和动态的数据光标。

在静态数据光标模式中，数据一次性从数据库中读出，并且数据以快照记录集合的形式给出。由于数据是一个快照(某个时间点上数据的一个静态视图)，所以数据源中其它用户对数据所做的修改是不可见的。动态数据光标通过保持现场数据，解决了这一问题，但是这要降低网络的传输能力和应用程序的性能。

键值驱动的数据光标介于上述两种模式之间，在这种模式中数据行在提取时就被标识，因此对于数据的修改可以进行跟踪。键值驱动数据光标对于实现向后滚动数据光标非常有用。在键值驱动数据光标中，在刷新之前添加和删除的全部记录行是不可见的。当进行向后滚动时，如果数据发生了任何改动，驱动程序会提取新的数据行。

注意：ODBC 也支持一种修改过的模式，在该模式中只提取键值的一个小窗口称之为混合数据光标(mixed cursor)，它对数据窗口使用键值驱动数据光标，对其余的数据使用动态数据光标。换句话说，数据窗口(称为一个行集合 RowSet)中的数据是键值驱动的，当访问窗口之外的数据时，使用动态模式提取另一个键值驱动的缓冲区。

数据光标应用程序 我们可能不知道在什么地方使用这些数据光标模式，并且为什么需要这些复杂的模式。简而言之，在信息系统中所有的数据光标模式都有其用武之地。

静态数据光标提供了数据的一个稳定视图，因为这些数据并不变化。这种模式非常适合于数据提炼和数据仓库式存放等类型的系统。对于这些应用程序，需要其数据稳定，以便可执行的信息系统产生出报告，或者以便于统计或分析使用。另外，静态数据光标对于大批量的数据检索，其执行情况也要优于其它模式。

另一方面，对于在线的订购系统或预定系统，则需要一个数据行锁定的系统动态视图，并且能观察到其它用户对数据的改动。在这种情况下，可使用动态数据光标。在许多这样的应

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用程序中，数据传输量较小，数据的访问是基于行逐行地执行的。对于这些在线应用程序，大块的数据访问是很罕见的。

书签(Bookmark)　书签是一个与数据光标模型有关的概念，但并不依赖于所使用的数据光标模式。书签是一个表中一个数据行的位置保存器。应用程序请求底层的数据库管理系统为一个数据行设定书签。DBMS 通常返回一个 32 位的标记，应用程序可以在以后使用这一标记得到这一行的数据。在 ODBC 中，我们可以使用 SQL_FETCH_BOOKMARK 选项调用 SQLExtendedFetch 函数获得一个书签。书签对于提高 GUI 应用程序的性能是很有用的，特别是对于使用一个类似于数据表界面的方式观察数据的应用程序。

定位更新/删除　这是一个与数据光标有关的概念，如果一个数据光标模型支持定位 UPDATE/DELETE，则我们可以在一个结果集(ResultSet)中更新/删除当前数据行，而无需更多的处理，例如锁定、读及提取等。

在 SQL 中，定位 UPDATE 或 DELETE 语句的形式为：

UPDATE/DELETE <字段或列值等> WHERE CURRENT OF <数据光标名>

定位 UPDATE 语名更新当前数据行的字段：

UPDATE <表> SET <字段> = <值> WHERE CURRENT OF <数据光标名>

定位 DELETE 语句删除当前数据行使用的格式为：

DELETE <表> WHERE CURRENT OF <数据光标名>

一般来说，对于这种类型的 SQL 语句，底层驱动程序或 DBMS 必须支持可更新性、并发性及动态滚动数据光标。但是，在应用程序层上，还有许多其它提供定位 UPDATE/DELETE 功能的方法。目前，JDBC 不支持任何一种高级的数据光标功能。然而，随着 JDBC 驱动程序实现的改进，相信 JDBC API 将会提供非常复杂的数据光标管理方法。

38.4.4　复制

数据复制是共同数据在组织内部许多位置上的分发。它提供了可靠性、容错性以及由于减少了通信而带来的数据访问性能，在许多情况下，它还提供了由于数据可以作为子集管理而带来的可管理性。

正如已经看到的，客户/服务器系统跨越一个组织，可能还包括它的客户和供应商——很可能是在一个很广的地理范围上。当谈论关于面向任务的应用程序时，特别是谈论关于现今全球商务市场时，系统跨越整个地球并非不常见的事情。如果所有的数据都集中在一个中央位置，要使系统有效地访问数据并提供很高的性能是几乎不可能的。同样，如果数据集中存放，在一个面向任务的系统中，一个单一的错误就会使整个商务系统瘫痪。所以在一个组织的各个地理位置上复制数据是一个很好的策略。

不同的厂商处理复制时的方式也不同。例如，Lotus Notes 群件产品所使用的复制模式是，把数据库看作是一个点，添加/更新/删除等操作在这些数据库之间传递。Lotus Notes 有一些复制公式，这些公式按照各种准则选择数据的子集进行复制。

而微软的 SQL Server 则使用一种发布者/预订者的模式进行复制。在这种模式中一个数据库或一个数据库的一部分可以向许多预订者发布。一个数据库可以是一个发布者也可以是一个预订者。例如，西部地区可以在从其它地区接收(预定)销售数据的同时发布自己的销售数据。

来自其它厂商的管理和非中心化数据的复制模式还有许多，复制是一种新技术，它正在慢慢地寻找其进入许多其它产品的途径。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 39 章　JDBC:Java 数据库连接

本章内容:

39.1　JDBC 概述
39.2　JDBC 的实现
39.3　Connection 类
39.4　元数据函数
39.5　SQLException 类
39.6　SQLWarning 类

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

39.1　JDBC 概述

JDBC 是 Java 数据库连接(Java Database Connectivity)API，它是来自于 Sun Microsystem 公司的 Java 企业版 API 的一部分。从开发人员的角度来看，JDBC 是标准化地将关系型数据库和 Java 程序集成到一起的第一个尝试。JDBC 打开了所有的关系功能，使其能够集成到 Java 小程序和应用程序中，在本章和下一章中，我们将对 JDBC 的类和方法进行详细介绍。

Java 数据库连接(JDBC)是用于与 SQL 数据源进行交互的关系型数据库对象和方法的集合。JDBC API 是 Java1.1 企业版 API 的一部分，所以它也是所有 Java 虚拟机(JVM)实现的一部分。

提示：尽管对象和方法是基于关系型数据库模型的，JDBC 并没有对底层数据源或数据存储模式做任何假设，用户可以使用 JDBC API 访问和检索来自许多数据源的音频或视频数据，并将其加载到 Java 对象中。唯一的要求是必须有针对那种数据源的 JDBC 实现。

Sun 公司于 1996 年 3 月份引入 JDBC API 规范，作为 0.50 版的草案，并公布于公众。该规范从 0.50 版开始，经过 1.0 版、1.10 版到现在的 1.22 版。JDK 1.1 中包括 JDBC。所以用户无需单独卸载 JDBC。JDBC 1.22 版规范可以在 http://java.sun.com/products/jdbc/中找到，其中包括根据厂商、开发人及一般公众的意见所做的所有改进。

注意：JDBC 的 Web 站点上有四个与 JDBC 规范有关的重要文档，分别是 JDBC 规范(jdbc.spec－0122.pdf)、JDBC API 文档 Part Ⅰ－JDBC 接口(jdbc.api.1－0122.pdf)、JDBC API 文档 Part Ⅱ——类和异常(Classes and Exceptions)(jdbc.api.2－0122.pdf)以及 JDK 1.1 的文档(jdbc.pdf)JDBC Guide：Getting Started。

现在让我们来看看起源和设计思想。JDBC 设计者以 X/Open SQL Call Level Interface (CLI)作为 API 的基础。并非巧合，ODBC 也是基于 X/Open CLI 设计的。Sun 公司的工程师们想从现有的 ODBC 实现和开发经验中获得借鉴，这样就使得独立软件开发商(ISV)和系统开发人员比较容易接受 JDBC。但是 ODBC 是对数据库管理系统(DBMS)的 C 接口，并不容易转换到 Java 上来。所以，JDBC 的设计在思想上沿袭了 ODBC，同时在其主要抽象和 SQL CLI 实现上也沿袭了 ODBC，将 SQL CLI 实现为“一个与 Java 系统的其余部分协调的接口”，JDBC 规范的 2.4 一节中描述了这一点。例如，代替 ODBC 使用 SQLBindColumn 和 SQLFetch 从结果中获得列值，JDBC 使用了更简单的方法(我们将在本章的稍后看到)。

39.1.1　JDBC 是如何工作的

正如前面所讨论的，JDBC 是在 CLI 模型上开发的。JDBC 定义了一套 API 对象和方法，用于与底层的数据库交互。一个 Java 程序首先打开与数据库的一个连接，生成一个语句对象，使用该 Statement 语句对象传送 SQL 语句到底层的 DBMS，并检索结果以及关于结果的信息。一般来说，JDBC 类文件和 Java 小程序驻留在客户端。它们也可以从网络上卸载而来。为了最小化执行期间的延迟时间，最好将 JDBC 类放在客户端。而 DBMS 和数据源一般则位

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

于一个远程服务器上。

图 39.1 给出了 JDBC 通信层的两种选择。Java 小程序与 JDBC 层在客户系统进行通信，与数据库在网上的交互则由驱动程序管理。

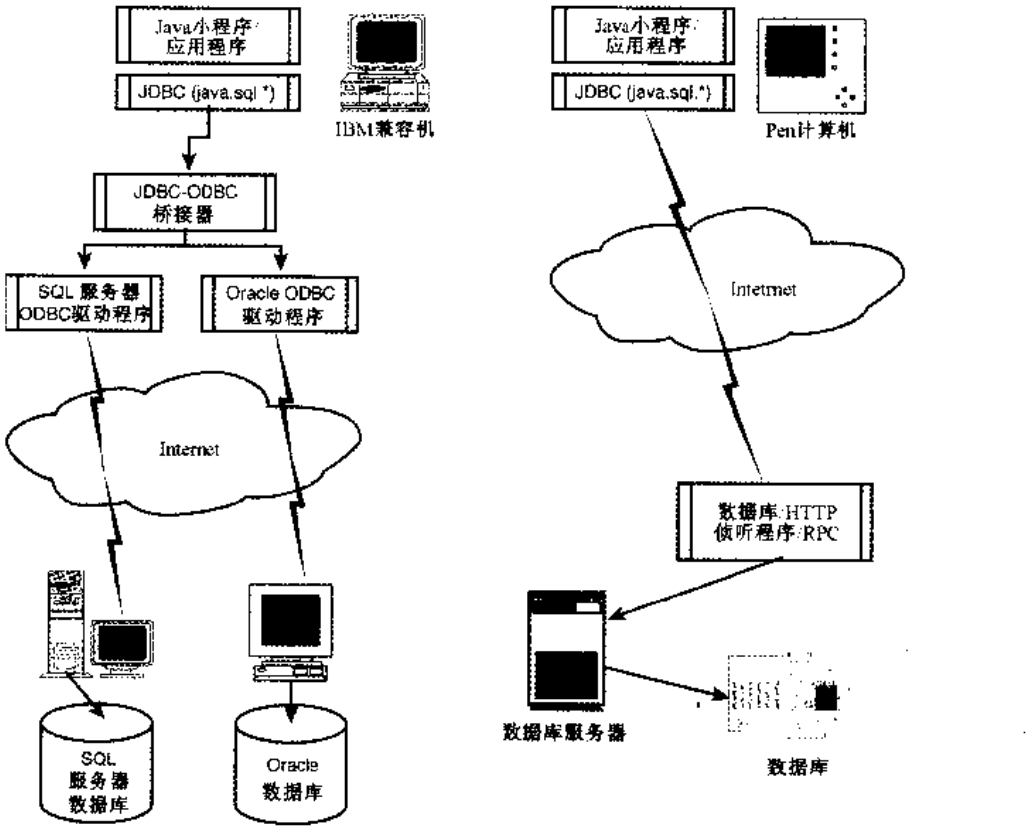


图 39.1 JDBC 通信层的两种选择：JDBC 驱动程序可以是一个本地库，与 JDBC－ODBC 桥接器一样；也可以是在网上与数据库服务器中的一个 RPC 或 HTTP 侦听进程交谈的一个 Java 类

JDBC 类在 java.sql 包内，所有的 Java 程序都使用 java.sql 包内的对象和方法从数据源读取或向数据源写入。使用 JDBC 的程序需要一个针对于那种要与其进行交互的数据源的驱动程序。该驱动程序可以是一个本地模块(如 Sun/Intersolv 开发的 Windows JDBC－ODBC 桥接器 JDBCODBC.DLL)，也可以是一个 Java 程序，该程序通过使用一些 RPC 或一个 HTTP 发送者——侦听者协议，在网络上与一个服务器交谈。两种模式均在图 39.1 中给出。

可以想象一个应用程序将要处理多个数据源的情况——还可能是不同种类的数据源。(数据库网关程序就是一个应用程序访问多个不同种类数据源的很好的例子。)出于这种原因，JDBC 提供了一个 DriverManager(驱动程序管理器)，其功能是管理驱动程序并向应用程序提供一个当前所加载的驱动程序清单。

数据源、数据库、还是数据库管理系统(DBMS)

尽管“数据库(database)”这一单词在 JDBC(Java Database Connectivity)的名字中出现，但它的形式，内容以及数据的位置对使用 JDBC 的 Java 程序都是不重要的，只要有针对于那种数据的驱动程序就可以。所以使用数据源这一术语来描述数据要比使用数据库、数据库管理系统(DBMS)、DB 或者文件更精确。将来 Java 设备如电视、问询机或网络计算机等，将访问、检索并操作来自于各种根本不是关系型数据库数据源的不同类型数据(音频、视频、图像、时间序列等等)。大部分的数据甚至可能不是来自于大量存储的数据。例如数据可能是来自于卫星

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的视频流,或来自于一个电话的音频流。

在使用一般性的术语描述时,ODBC 也使用数据源这一术语而不是数据库。

39.1.2　安全模型

安全性总是一个重要的问题,特别是牵涉到数据库的时候。正如本书中所写到的,JDBC 遵循了标准的安全模型,Java 小程序只能连接到下载它们的服务器;远程小程序不能连接到本地数据库。而对于应用程序没有连接限制。对于纯 Java 驱动程序,安全检查是自动的。而对使用本地方法开发的驱动程序则必须有一些安全性检查。

> **注意**:使用 Java 1.1 和 Java 安全性 API,我们可以建立"信任的关系",使用这种关系可以检查可信任的站点。然后,我们可以给予从可信任源下载的 Java 小程序更多的功能,包括给予访问本地资源的权限。关于 Java 安全性更详细的介绍,参见第 34 章"Java 安全性深析"。

39.1.3　JDBC – ODBC 桥接器

作为 JDBC 的一部分,Sun 公司还发行了一个驱动程序,用于从 JDBC 访问 ODBC 数据源。该驱动程序是 Sun 公司与 Intersolv 公司联合开发的,称之为 JDBC – ODBC 桥接器。JDBC – ODBC 桥接器的实现是 JdbcOdbc.Class 和一个用于访问 ODBC 驱动程序的本地库。对于 Windows 平台该本地库是一个动态连接库 DLL(JDBCODBC.DLL)。

由于 JDBC 在设计上与 ODBC 很接近,ODBC 桥接器是在 JDBC 上的一个较薄的层。在内部,这个驱动程序把 JDBC 的方法映射到 ODBC 调用上,这样,就可以和任何可用的 ODBC 驱动程序进行交互了。这种桥接器的优点是,它使 JDBC 目前有能力访问几乎所有的数据库,因为 ODBC 驱动程序被广泛使用,是很容易得到的。我们可以使用这种桥接器(版本 1.2001)运行本章和下一章中的样本程序。

39.2　JDBC 实现

JDBC 的实现是 java.sql 包。该软件包包括所有的 JDBC 类和方法,见表 39.1。

表 39.1　JDBC 类

类型	类
驱动程序	java.sql.Driver java.sql.DriverManager java.sql.DriverPropertyInfo
连接	java.sql.Connection
语句	java.sql.Statement java.sql.PreparedStatement java.sql.CallableStatement

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

类型	类
结果集	java.sql.ResultSet
错误/警告	java.sql.SQLException java.sql.SQLWarning
元数据	java.sql.DatebaseMetaData java.sql.ResultSetMetaData
日期/时间	java.sql.Date java.sql.Time java.sql.Timestamp
其它	java.sql.Types java.sql.DataTruncation

现在让我们来看看这些类,以及如何开发一个简单的 JDBC 应用程序。

39.2.1 JDBC 类——概述

当我们查看 JDBC 类的层次关系及与之相关的方法时,层次关系中最上层的类是 DriverManager(驱动程序管理器)。DriverManager 类中有驱动程序信息等。当每个驱动程序被加载时,它要使用 DriverManager 进行注册。当需要打开一个连接时,DriverManager 根据 JDBC 的 URL 选择驱动程序。

JDBC 的 URL

与 Internet 一样,JDBC 识别一个数据库时也使用 URL。URL 的形式如下所示:

jabc: < 子协议(subprotocol) > : < 与 DBMS/Protocol 相关的子名字 >

对于 Internet 或 Intranet 上的数据库,子名字(subname)包括网络的 URL//hostname:port/。 < 子协议(subprotocol) > 可以是任何一个数据库可以理解的名字。odbc 子协议名被保留下来用于 ODBC 风格的数据源。对一般的 ODBC 数据库,JDBC URL 如下表示:

jdbc:odbc: < ODBC DSN > ;User = < 用户名 > ;PW = < 口令字 >

如果我们正在使用一个新的子协议开发一个 JDBC 驱动程序,最好是与 Sun 一起保留该子协议的名字,Sun 维护着一个非正式的子协议登记。

引用 java.sql.Driver 类通常用来获得 PropertyInfo(属性信息)、版本号等信息。在执行一个使用 JDBC API 的 Java 程序时,这个类可以加载多次。

请看表 39.2 中所列出的 java.sql.Driver 和 java.sql.DriverManager 类和方法。可以看出,当我们使用 getConnection( ) 方法时,DriverManager 返回一个 Connection 对象。

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

表 39.2　Driver、DriverManager 类及相关方法

返回类型	方法名称	参数
Java.sql.Driver 类		
Connection	connect	(String url, java.util.Properties info)
boolean	acceptsURL	(String url)
DriverPropertyInfo[ ]	getPropertyInfo	(String url, java.util.Properties info)
int	getMajorVersion	()
int	getMinorVersion	()
boolean	jdbcCompliant	()
java.sql.DriverManager 类		
Connection	getConnection	(String url, java.utl.Properties info)
Connection	getConnection	(String url, String user, String password)
Connection	getConnection	(String url)
Driver	getDriver	(String url)
void	registerDriver	(java.sql.Driver driver)
void	deregisterDriver	(Driver driver)
java.util.Enumeration	getDrivers	()
void	setLoginTimeout	(int seconds)
int	getLoginTimeout	()
void	setLogStream	(java.io.Print Stream.out)
java.io.PrintStream	getLogStream	()
void	println	(String message)
类初始化程序		
void	initialize	()

其它有用的方法包括 registerDriver()、deRegister()以及 getDrivers()等方法，通过使用 getDrivers()方法，我们可以得到一个已登记的驱动程序列表。图 39.2 给出了 JDBC 类的层次关系，以及典型的使用 JDBC API 的 Java 程序流程。

在下面的一节中，我们将按照使用 JDBC 和 JDBC－ODBC 驱动程序访问一个简单数据库的步骤进行介绍。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

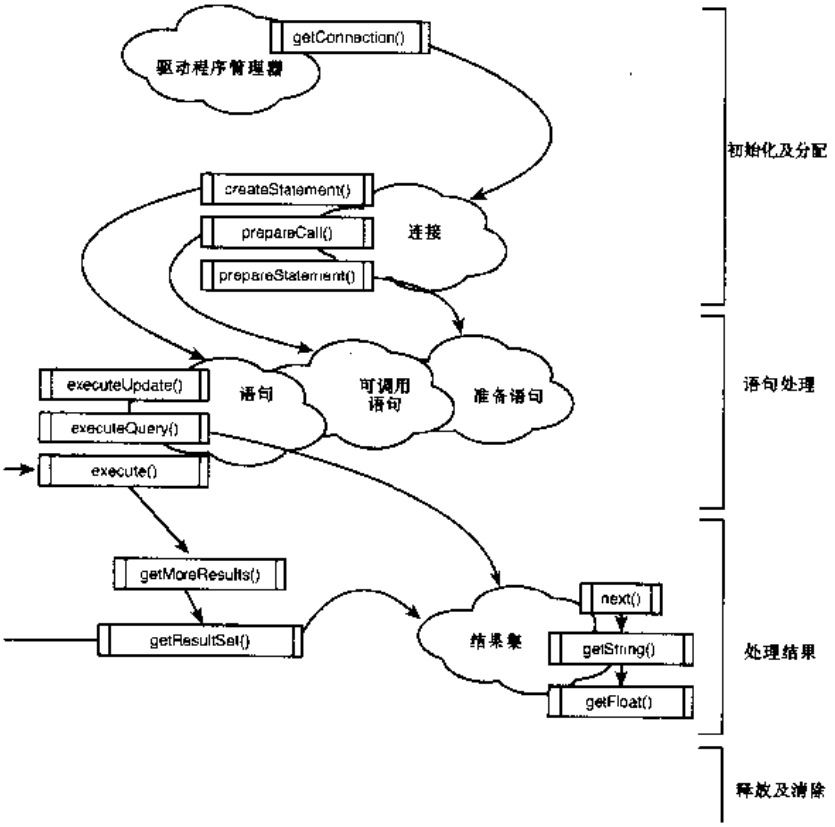


图 39.2 JDBC 类的层次关系及 JDBC API 流程

39.2.2 JDBC 应用程序剖析

为了处理来自于一个数据库中的数据,Java 程序采用以下这些一般性步骤。(图 39.2 中给出了一般的 JDBC 对象、方法及顺序)。第一步,程序调用 getConnection()方法,获得一个 Connection 对象。然后,程序创建一个 Statement 对象并准备一个 SQL 语句。

一个 SQL 语句可以是一个被立即执行的语句(Statement 对象),可以是一个被编译的语句(PreparedStatement 对象),也可以是一个存储过程的调用(CallableStatement 对象)。当 executeQuery()方法执行时,返回一个 ResultSet 对象。像 update 或 delete 这样的 SQL 语句不返回 ResultSet 对象。对这种语句,要使用 executeUpdate()方法。executeUpdate()方法返回一个整数,它表示 SQL 语句所影响到的数据行数。

ResultSet 中包含有数据行,使用 next()方法,可以对其进行分析。在一个事务处理应用程序中,像 rollback()和 commit()这样的方法,可以用来对 SQL 语句所做的改动进行取消,或使 SQL 语句所做的改动永久生效。

39.2.3 JDBC 实例

本节中的这些例子访问的是 Student 数据库,其数据库设计图见图 39.3。我们所感兴趣的这个例子中的表包括 Student (学生)表、Classes (班级)表、Instructors(教师)表以及 Student - Classes(学生班级)表。这一数据库是一个 Microsoft Access 数据库。全部数据库及样本数据都是由 Access 数据库向导产生的。我们可以使用 JDBC 和 JDBC - ODBC 桥接器来访问该数据库。

在开始编写 Java JDBC 程序之前,我们需要配置一个 ODBC 数据源。正如前面所看到的,

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

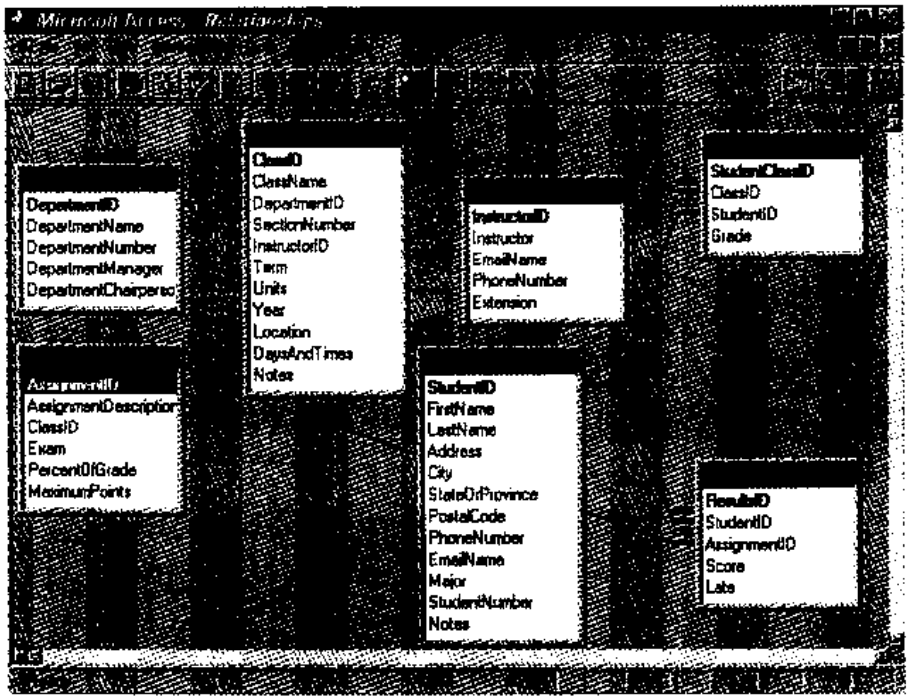


图 39.3　JDBC 实例数据库设计图

getConnection()方法需要 ODBC 数据源的一个数据源名字(DNS)、用户 ID 和口令字。数据库驱动程序类型或子协议名是 odbc。驱动程序管理器从 ODBC 驱动程序中找出其它的细节。

但是，其它的细节放在哪里呢？这正是 ODBC 设置要解决的问题。ODBC 的设置程序在 Java 应用程序之外，从微软的 ODBC 程序组中运行。我们可以使用 ODBC 设置程序设置数据源，以便使 ODBC 驱动程序管理器能够得到这种信息。接着 ODBC 驱动程序管理器加载 Microsoft Access ODBC 驱动程序。如果上述数据库在其它的 DBMS 如 Oracle 中，则我们将配置这一数据源为 Oracle ODBC 驱动程序。在 Windows 3.x 中，设置程序(Setup)将这些信息存放在 ODBC.INI 文件中，在 Windows 95 和 Windows NT4.0 中，这些信息存放在注册表(Registry)中。图 39.4 给出了 ODBC 设置的屏幕显示。

JDBC 查询实例　在这一实例中，我们使用一个 SQL SELECT 语句列出数据库中所有学生的清单。使用 JDBC API 完成这一任务所需要的步骤如下所示。对每一个步骤，使用 JDBC API 调用的 Java 程序代码都遵循对这些步骤的描述。

```
//声明一个方法和一些变量
public void ListStudents() throws SQLException {
    int i, NoOfColumns;
    String StNo,StFName,StLName;
    //初始化并加载 JDBC - ODBC 驱动程序
    Class.forName("jdbc.odbc.JdbcOdbcDriver");
    //创建连接对象
    Connection Ex1Con = DriverManager.getConnection("jdbc.odbc:StudentDB";uid = "admin";pw = "sa");
    //创建一个简单的 Statement 对象
    Statement Ex1Stmt = Ex1Con.createStatement();
    //创建 SQL 串，传送到 DBMS 并执行 SQL 语句
    ResultSet Ex1rs = Ex1Stmt.executeQuery(
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

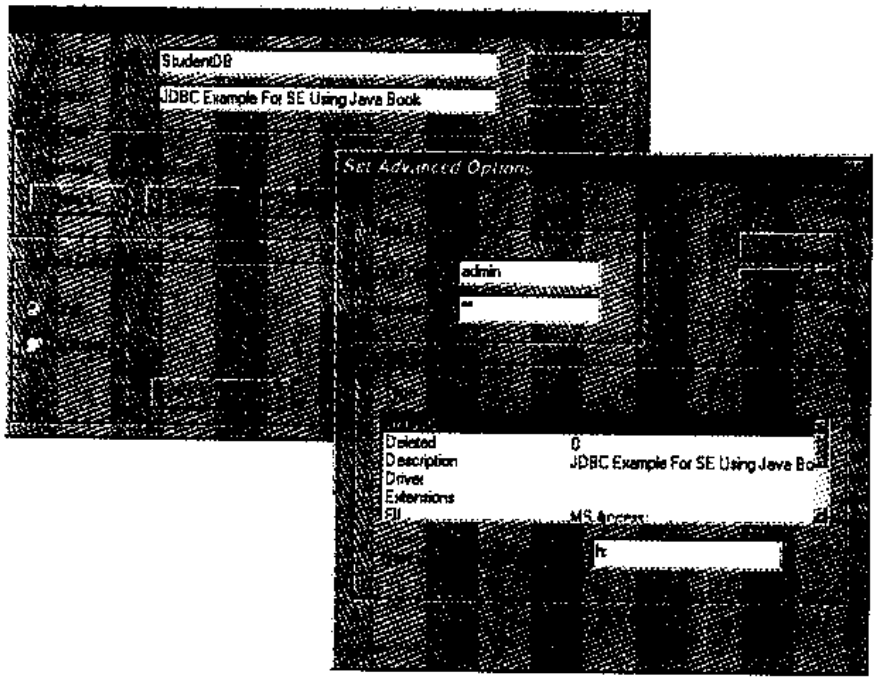


图 39.4 为数据库实例进行 ODBC 设置。经过设置后,数据库实例的 URL 是 jdbc:odbc:Student DB;uid="admin";pw="sa"

```
        "SELECT StudentNumber, FirstName, LastName FROM Students");
    //处理每一个数据行直到不再有数据行
    //在控制台显示结果
    System.out.println("Student Number    First Name    Last Name");
    while (Ex1rs.next()) {
        //将列值保存到 Java 变量中
        StNo = Ex1rs.getString(1);
        StFName = Ex1rs.getString(2);
        StLName = Ex1rs.getString(3);
        System.out.println(StNo,StFName,StLName);
    }
}
```

可以看出,这是一个使用 JDBC API 的简单 Java 程序。该程序演示了访问一个表并列出记录中某些字段值所需的基本步骤。

JDBC 更新实例 在这一实例中,我们根据已知的学生的 StudentNumber,更新 Student 表中 FirstName 字段的值。与上述实例一样,代码遵循对每个步骤的描述。

```
//声明一个方法和一些变量及参数
public void UpdateStudentName(String StFName, String StLName,
    String StNo) throws SQLException {
    int RetValue;
    //初始化并加载 JDBC-ODBC 驱动程序
    Class.forName ("jdbc.odbc.JdbcOdbcDriver");
    //创建连接对象
    Connection Ex1Con = DriverManager.getConnection("jdbc:odbc:StudentDB";uid="admin";pw="sa");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
//创建一个简单的 Statement 对象
Statement Ex1Stmt = Ex1Con.createStatement();
//创建 SQL 串,传送到 DBMS 并执行该 SQL 语句
String SQLBuffer = "UPDATE Students SET FirstName = " +
    StFName + ",LastName = " + StLName +
    " WHERE StudentNumber = " + StNo;
RetValue = Ex1Stmt.executeUpdate(SQLBuffer);
System.out.println("Updated " + RetValue + " rows in the Database.");
}
```

在该例中,我们使用 SQL 语句并从 DBMS 中得到 SQL 语句所作用到的数据行行数。

前面的两个实例给出了如何在 Java 程序中使用 JDBC API,对底层数据进行简单但功能强大的 SQL 操作。在下面的几节中,我们将详细考察 JDBC 中的每一个类。

39.3　Connection 类

Connection 类是 JDBC 中的主要类之一。它包含了许多功能,范围从事务处理到创建语句等,见表 39.3。

表 39.3　java.sql.Connection 类的方法和常量

返回类型	方法名称	参数
和语句相关的方法		
Statement	createStatement	()
PreparedStatement	prepareStatement	(String sql)
CallableStatement	prepareCal	(String sql)
String	nativeSQL	(String sql)
void	close	()
boolean	isClosed	()
和元数据相关的方法		
DatabaseMetaData	getMetaData	()
void	setReadOnly	(boolean readOnly)
boolean	isReadOnly	()
void	setCatalog	(String catalog)
String	getCatalog	()
SQLWarning	getWarnings	()
void	clearWarnings	()
和事务相关的方法		
void	setAutoCommit	(boolean autoCommit)
boolean	getAutoCommit	()
void	commit	()
void	rollback	()
void	setTransaction Isolation	(int level)
int	getTransaction Isolation	()

TransactionIsolation 常量为整型数,定义在 java.sql.Connection 中,如下所示:

TransactionIsolation 常量名　　值

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

TRANSACTION-NONE	0
TRANSACTION-READ-UNCOMMITTED	1
TRANSACTION-READ-COMMITTED	2
TRANSACTION-REPEATABLE-READ	4
TRANSACTION-SERIALIZABLE	8

正如我们在前面所看到的,这个连接是针对一个可以使用一种特定的子协议与之交互的特定数据库的。Connection 对象在内部管理关于连接的各个方面,这些细节对于程序是透明的。实际上,Connection 对象是一个通往底层 DBMS 驱动程序的管道线。被管理的信息包括数据源标识、子协议、状态信息、DBMS 执行计划 ID 或句柄以及与底层 DBMS 成功地交互所需的其它上下文信息。

注意:数据源标识可以是 Internet 数据库服务器的一个端口,使用//< server name >:port/URL 进行标识;可以只是 ODBC 驱动程序所使用的数据源的名字;也可以是本地计算机上一个数据库文件的完整路径。就我们所知,它可以是华尔街股票市场馈给的一个指针。

另一个由 Connection 对象所执行的重要功能是事务管理。事务处理依赖于一个内部自动提交标志的状态,该标志使用 setAutoCommit()方法进行设置,该标志的状态可使用 getAutoCommit()方法读取。当该标志为 true(真)值时,一旦事务处理完成,则自动地被提交,不需要 Java 应用程序进行任何干预或命令。当该标志为 false(假)值时,系统处于手工方式。Java 程序可以进行选择:使用 Commit() 方法提交自上次提交以后所发生过的一组事务,或者使用 rollback()方法回退所发生的事务。

注意:JDBC 还提供设置事务独立模块化的方法。当我们开发多层应用程序时,多个用户将会在同一数据库表上并发地执行相互交织的事务。数据库驱动程序必须采用复杂的锁定机制,数据缓冲区算法,以及大型 JDBC 应用程序所需的实现事务独立性的机制。当多个 Java 对象工作于可能分布在全球各地的许多数据库时,这种情况更为复杂。只有时间能够告诉我们,在将来的 Internet/Intranet 中,对于事务独立性的特殊需要将会是什么。

当我们有了对一个数据源的一个成功的 Connection 对象时,可以有很多方式与数据源进行交互。从应用程序开发人员的角度看,最常用的方法是使用处理 SQL 语句的对象。在 JDBC 中,主要有三种类型的语句:

- Stastement
- PreparedStatement
- CallableStatement

Connection 对象中有 createStatement() 、preparedStatement() 以及 PrepareCall()方法,分别用来创建这些语句对象。在第 40 章"JDBC 考察"中将详细介绍语句类型对象。

39.4 元数据函数

从理论上讲,元数据(Metadata)是关于数据的信息。Metadata 方法的主要目标是数据库

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

工具和需要关于底层 DBMS 的功能和结构信息的向导(wizard)。大多数情况下,这些工具需要 SQL 语句所返回的结果集的动态信息。JDBC 有两个元数据类:ResultSetMetaData 和 DatabaseMetaData。正如在方法表中所看到的,在这两个类中有非常非常多的方法可以使用。

39.4.1　DatabaseMetaData

DatabaseMetaData 类似于 ODBC 中的分类函数,应用程序用它查询底层 DBMS 的系统表并获取信息。ODBC 以结果集的形式返回信息。JDBC 返回的结果则是具有完善定义列的一个 ResultSet 对象。

DatabaseMetaData 对象及其方法给出了关于底层数据库的大量信息。这些信息对于数据库工具、自动数据转换以及网关程序尤为有用。表 39.4 给出了所有 DatabaseMetaData 对象的方法。正如所看到的,它是一个很长的表,有一百多个方法。除非是功能非常全面的 GUI 工具,大多数的程序并不会使用全部这些方法。但是,作为一个开发人员,总会有需要知道关于数据库的某些特点或了解是否支持某个特性这样的时刻,也正是在这样的时刻有了下面的表就很方便了。

表 39.4　DatabaseMetaData 方法

返回类型	方法名称	参数
boolean	allProceduresAreCallable	( )
boolean	allTablesAreSelectable	( )
String	getURL	( )
String	getUserName	( )
boolean	isReadOnly	( )
boolean	nullsAreSortedHigh	( )
boolean	nullsAreSortedLow	( )
boolean	nullsAreSortedAtStart	( )
boolean	nullsAreSortedAtEnd	( )
String	getDatabaseProductName	( )
String	getDatabaseProductVersion	( )
String	getDriverName	( )
String	getDriverVersion	( )
int	getDriverMajorVersion	( )
int	getDriverMinorVersion	( )
boolean	usesLocalFiles	( )
boolean	usesLocalFilePerTable	( )
boolean	supportsMixedCaseIdentifiers	( )

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

返回类型	方法名称	参数
boolean	storesUpperCaseIdentifiers	( )
boolean	storesLowerCaseIdentifiers	( )
boolean	storesMixedCaseIdentifiers	( )
boolean	supportsMixedCaseQuoted Identifiers	( )
boolean	storesUpperCaseQuoted Identifiers	( )
boolean	storesLowerCaseQuoted Identifiers	( )
boolean	storesMixedCaseQuoted Identifiers	( )
String	getIdentifierQuoteString	( )
String	getSQLKeywords	( )
String	getNumericFunctions	( )
String	getStringFunctions	( )
String	getSystemFunctions	( )
String	getTimeDateFunctions	( )
String	getSearchStringEscape	( )
String	getExtraNameCharacters	( )
boolean	supportsAlterTableWithAdd Column	( )
boolean	supportsAlterTableWithDrop Column	( )
boolean	supportsColumnAliasing	( )
boolean	nullPlusNonNullIsNull	( )
boolean	supportsConvert	( )
boolean	supportsConvert	(int fromType,int toType)
boolean	supportsTableCorrelationNames	( )
boolean	supportsDifferentTable CorrelationNames	( )
boolean	supportsExpressionsInOrderBy	( )
boolean	supportOrderByUnrelated	( )
boolean	supportsGroupBy	( )
boolean	supportsGroupByUnrelated	( )
boolean	supportsGroupByBeyondSelect	( )
boolean	supportsLikeEscapeClause	( )
boolean	supportsMultipleResultSets	( )
boolean	supportsMultipleTransactions	( )
boolean	supportsNonNullableColumns	( )
boolean	supportsMinimumSQLGrammar	( )
boolean	supportsCoreSQLGrammar	( )
boolean	supportsExtendedSQLGrammar	( )
boolean	supportsANSI92EntryLevelSQL	( )
boolean	supportsANSI92IntermediateSQL	( )

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

返回类型	方法名称	参数
boolean	supportsANSI92FullSQL	( )
boolean	supportsIntegrityEnhancementFacility	( )
boolean	supportsOuterJoins	( )
boolean	supportsFullOuterJoins	( )
boolean	supportsLimitedOuterJoins	( )
String	getSchemaTerm	( )
String	getProcedureTerm	( )
String	getCatalogTerm	( )
boolean	isCatalogAtStart	( )
String	etCatalogSeparator	( )
boolean	supportsSchemasInDataManipulation	( )
boolean	supportsSchemasInProcedureCalls	( )
boolean	supportsSchemasInTableDefinitions	( )
boolean	supportsSchemasInIndexDefinitions	( )
boolean	supportsSchemasInPrivilegeDefinitions	( )
boolean	supportsCatalogsInDataManipulation	( )
boolean	supportsCatalogsInProcedureCalls	( )
boolean	supportsCatalogsInTableDefinition	( )
boolean	supportsCatalogsInIndexDefinitions	( )
boolean	supportsCatalogsInPrivilegeDefinitions	( )
boolean	supportsPositionedDelete	( )
boolean	supportsPositionedUpdate	( )
boolean	supportsSelectForUpdate	( )
boolean	supportsStoredProcedures	( )
boolean	supportsSubqueriesInComparisons	( )
boolean	supportsSubqueriesInExists	( )
boolean	supportsSubqueriesInIns	( )
boolean	supportsSubqueriesInQuantifieds	( )
boolean	supportsCorrelatedSubqueries	( )
boolean	supportsUnion	( )
boolean	supportsUnionAll	( )
boolean	supportsOpenCursorsAcrossCommit	( )
boolean	supportsOpenCursorsAcrossRollback	( )
boolean	supportsOpenStatementsAcrossCommit	( )
boolean	supportsOpenStatementsAcrossRollback	( )
int	getMaxBinaryLiteralLength	( )
int	getMaxCharLiteralLength	( )

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

返回类型	方法名称	参数
int	getMaxColumnNameLength	( )
int	getMaxColumnsInGroupBy	( )
int	getMaxColumnsInIndex	( )
int	getMaxColumnsInOrderBy	( )
int	getMaxColumnsInSelect	( )
int	getMaxColumnsInTable	( )
int	getMaxConnections	( )
int	getMaxCursorNameLength	( )
int	getMaxIndexLength	( )
int	getMaxSchemaNameLength	( )
int	getMaxProcedureNameLength	( )
int	getMaxCatalogNameLength	( )
int	getMaxRowSize	( )
boolean	doesMaxRowSizeIncludeBlobs	( )
int	getMaxStatementLength	( )
int	getMaxStatements	( )
int	getMaxTableNameLength	( )
int	getMaxTablesInSelect	( )
int	getMaxUserNameLength	( )
int	getDefaultTransactionIsolation	( )
boolean	supportsTransactions	( )
boolean	supportsTransaction IsolationLevel	(int level)
boolean	supportsDataDefinitionAndDataManipulation Transactions	( )
boolean	supportsDataManipulationTransactionsOnly	( )
boolean	dataDefinitionCausesTransactionCommit	( )
boolean	dataDefinitionIgnoredInTransactions	( )
ResultSet	getProcedures	(String catalog, String schemaPattern, String procedureNamePattern)
ResultSet	getProcedureColumns	(String catalog, String schemaPattern, String procedureNamePattern, StringcolumnName - Pattern)
ResultSet	getTables	(String catalog, StringschemaPattern, String tableNamePattern,String types[ ])
ResultSet	getSchemas	( )
ResultSet	getCatalogs	( )
ResultSet	getTableTypes	( )

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

返回类型	方法名称	参数
ResultSet	getColumns	(String catalog, StringschemaPattern, StringtableNamePattern, StringcolumnName - Pattern)
ResultSet	getColumnPrivileges	(String catalog, String schema, String table, String columnNamePattern)
ResultSet	getTablePrivileges	(String catalog, String schemaPattern, StringtableNamePattern)
ResultSet	getBestRowIdentifier	(String catalog, String schema, String table, int scope, boolean nullable)
ResultSet	getVersionColumns	(String catalog, String schema, String table)
ResultSet	getPrimaryKeys	(String catalog, String schema, String table)
ResultSet	getImportedKeys	(String catalog, String schema, String table)
ResultSet	getExportedKeys	(String catalog, String schema, String table)
ResultSet	getCrossReference	(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)
ResultSet	getTypeInfo	( )
ResultSet	getIndexInfo	(String catalog, String schema, String table, boolean unique, boolean approximate)

从表中可以看出，DatabaseMetaData 对象给出了关于底层 DBMS 的功能及限制的信息。对于应用程序编程人员非常有用的一组重要信息包括：描述数据库中表的设计细节、表名、存储的过程名等方法。

在 Java 应用程序中使用 DatabaseMetaData 对象的一个例子是多层次和可扩展的应用程序的开发。如果底层数据库引擎支持某一特别的特性，则 Java 应用程序可以查询；如果它不支持，那么 Java 可以选择调用另外的方法来执行这一任务。这样，如果一种特性在 DBMS 中是不可用的，应用程序也不会因此而失败。

同时，一旦高级的功能可以使用，应用程序将使用它，这就是有些专家所说的“内部可操作性和可扩展性”。内部可操作性对于应用程序工具也是需要的——尤其是对于必须与不同的数据源进行交互、基于 Java 的通用设计和查询工具。这些工具必须查询数据源系统，找出所支持的特性，并按照查询结果继续下去。对于支持高级特性的数据源，这些工具可能能够以更快的速度处理信息，对于特性丰富的数据源它们也许能够为用户提供更多的选择。

39.4.2　ResultSetMetaData

与 DatabaseMetaData 相比，ResultSetMetaData 对象要简单一些，所具有的方法也较少一些。但这些方法在应用程序开发人员中也更为流行。如同它的名字所暗示的那样，ResultSetMetaData 描述了一个 ResultSet 对象。表 39.5 中列出了 ResultSetMetaData 对象中所有可用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的方法。

表 39.5 ResultSetMetaData 方法

返回类型	方法名称	参数
Int	getColumnCount	()
boolean	isAutoIncrement	(int column)
boolean	isCaseSensitive	(int column)
boolean	isSearchable	(int column)
boolean	isCurrency	(int column)
int	isNullable	(int column)
boolean	isSigned	(int column)
int	getColumnDisplaySize	(int column)
String	getColumnLabel	(int column)
String	getColumnName	(int column)
String	getSchemaName	(int column)
int	getPrecision	(int column)
int	getScale	(int column)
String	getTableName	(int column)
String	getCatalogName	(int column)
int	getColumnType	(int column)
String	getColumnTypeName	(int column)
boolean	isReadOnly	(int column)
boolean	isWritable	(int column)
boolean	isDefinitelyWritable	(int column)
返回值		
int columnNoNulls = 0		
int columnNullable = 1		
int columnNullableUnknown = 2		

从表中可以看出，ResultSetMetaData 对象可用于查找一个结果集中的某个列的类型和属性。即使在普通的应用程序中，也需要使用像 getColumnLabel() 和 getColumnDisplaySize()这样的方法。使用这些方法将使得程序对结果集进行一般性处理，并保证一个组织内部的各种应用程序的统一性，因为所有名字和尺寸都是从数据库自身中取出的。

在结束本章前，还应该看看 JDBC 所提供的异常处理功能。

39.5 SQLException 类

JDBC 中的 SQLException 类提供了关于数据库访问期间所发生的错误的各种信息。SQLException 对象是用链链接起来的，所以程序可以顺序地对其进行读取。这是一个很好的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

机制，一个错误条件可能产生多个错误，最后一个错误可能与实际的错误条件并无关系。通过链接这些错误，我们可以真正地查明第一个错误。每一个 SQLException 都有一条错误信息和一个厂商指定的错误码。与一个 SQLException 有联系的是 SQLState 字符串，该字符串遵循 SQL 规范中所定义的 XOPEN SQLState 值。表 39.6 列出了 SQLException 类的所有方法。

表 39.6　SQLException 类的方法

返回类型	方法名称	参数
SQLException	SQLException	(String reason, String SQLState, int vendorCode)
SQLException	SQLException	(String reason, String SQLState)
SQLException	SQLException	(String reason)
SQLException	SQLException	()
String	getSQLState	()
int	getErrorCode	()
SQLException	getNextException	()
void	setNextException	(SQLException ex)

39.6　SQLWarning 类

与 SQLException 类不同，SQLWarnings 类并不引起 Java 程序的任何混乱。SQLWarnings 类附属于其方法引起该警告信息的对象上。所以我们可以使用 getWarnings( )方法检查警告信息，该方法可用于所有对象。表 39.7 列出了 SQLWarnings 类相关的所有方法。

表 39.7　SQLWarnings 的方法

返回类型	方法名称	参数
SQLWarning	SQLWarning	(String reason, String SQLState, int vendorCode)
SQLWarning	SQLWarning	(String reason, String SQLState)
SQLWarning	SQLWarning	(String reason)
SQLWarning	SQLWarning	()
SQLWarning	getNextWarning	()
void	setNextWarning	(SQLWarning w)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 40 章　JDBC 考察

本章内容：

40.1　语句
40.2　ResultSet 处理检查结果
40.3　其它 JDBC 类
40.4　JDBC 的前景

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

40.1　语句

Statement 对象根据 SQL 语句实现与数据库管理系统交互的所有工作。我们可以从一个 Connection 对象创建许多 Statement 对象。从内部来看，Statement 对象将存储与数据库交互所需的各种数据，包括状态信息、缓冲区句柄等，但对于 JDBC 应用程序来说这些内容是透明的。

> **注意**：当一个程序试图进行一个与系统内部状态不同步的操作时（例如，在没有 SQL 语句运行的情况下，试图使用 next()方法获得一个数据行），产生矛盾并引发异常。一般情况下这种异常由应用程序使用 SQLException 对象中的方法进行探测。

JDBC 支持三种类型的语句：

- Statement
- PreparedStatement
- CallableStatement

在开始探讨这些不同的语句之前，让我们先来看看一个 SQL 语句所要经过的步骤。

Java 应用程序首先在一个字符串缓冲区中创建一个 SQL 语句，并通过一些 API 调用将其传送到底层的 DBMS。SQL 语句在执行之前，需要进行语法检查、优化、并转换成为可执行的形式。在调用层接口（Call Level Interface，CLT）的应用程序接口（Application Program Interfare，API）模型中，应用程序通过驱动程序传送 SQL 语句到准备并执行 SQL 语句的底层数据库。

当 DBMS 接收到 SQL 字符串缓冲区后，DBMS 对语句进行分析并进行语法检查。如果语句的语法不正确，系统向驱动程序返回一个错误状态，由该驱动程序产生一个 SQLException 异常。如果 SQL 语句的语法正确，根据所使用的 DBMS，通常情况下产生许多查询计划，这些查询计划要通过一个优化器（经常是一个基于开销的优化器）进行优化。然后把这个经过优化的计划转换成一个二进制 SQL 语句版本的句柄或标识。

JDBC 三种类型的语句（Statement、PreparedStatement 和 CallableStatement）在 SQL 语句的准备和语句的执行时序上是不同的。对于简单的 Statement 对象，SQL 的准备和运行是在同一步中——至少从应用程序的角度看是这样。（从内部来看，驱动程序可能先要获得语句标识，然后命令 DBMS 执行查询，最后抛弃句柄。）对于 PreparedStatement 对象，驱动程序存储执行计划句柄以备以后使用。对于 CallableStatement 对象，SQL 语句实际上是进行一个通常已经优化了的存储过程的调用。

> **注意**：我们知道，存储过程是封装的商务规则程序，它们驻留在数据库服务器中。这些驻留的过程还加强了应用程序之间的统一性，并提供了数据库访问的安全性。最后，存储的过程独立于程序执行，所以应用程序不必花费时间等待 DBMS 创建执行计划。

现在让我们更仔细地观察一下每一种语句，以及这些语句提供给 Java 程序的东西。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

40.1.1 Statement

Statement 对象是使用 Connection 对象中的 createStatement()方法创建的。表 40.1 中给出了 Statement 对象中所有可使用的方法。

表 40.1 Statement 对象的方法

返回类型	方法名称	参数
ResultSet	executeQuery	(String sql)
int	executeUpdate	(String sql)
boolean	execute	(String sql)
boolean	getMoreResults	()
void	close	()
int	getMaxFieldSize	()
void	setMaxFieldSize	(int max)
int	getMaxRows	()
void	setMaxRows	(int max)
void	setEscapeProcessing	(boolean enable)
int	getQueryTimeout	()
void	setQueryTimeout	(int seconds)
void	cancel	()
java.sql.SQLWarning	getWarnings	()
void	clearWarnings	()
void	setCursorName	(String name)
ResultSet	getResultSet	()
int	getUpdateCount	()

这些方法中最重要的方法有 executeQuery()、executeUpdate()以及 execute()。当我们使用 SQL 语句创建 Statement 对象时，executeQuery()方法使用一个 SQL 字符串，通过驱动程序管理器将 SQL 字符串传送给底层的数据源，并且把 ResultSet 取回到应用程序。executeQuery()方法仅返回一个 ResultSet。对于返回多个 ResultSet 的情况，应该使用 execute()方法。

警告：一个 Statement 对象一次只能打开一个 ResultSet。

对于不返回 ResultSet 的 SQL 语句如 UPDATE、DELETE、以及 DDL 语句，Statement 对象有 executeUpdate()方法，它使用一个 SQL 字符串并返回一个整数。该整数表示 SQL 语句所影响到的数据行数。

> **注意**：JDBC 的处理是同步的，即应用程序必须等待 SQL 语句的完成。但是由于 Java 是一个多线程的平台，JDBC 设计者建议使用线程来模拟异步处理。

Statement 对象最适合于特定的 SQL 语句或立即被执行的 SQL 语句。一旦 DBMS 接收到该 SQL 语句，便对其进行语法分析、执行计划优化以及执行计划的生成等。然后 DBMS 执行查询，放弃优化的执行计划；所以如果再次调用 executeQuery()方法，则 DBMS 将再一次执行所有这些步骤。

下述的实例程序给出了如何使用 Statement 类访问数据库的例子。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

参见 39.2.2“JDBC 应用程序剖析”。

在程序清单 40.1 中所给出的例子中，我们列出了所有在登记数据库中可用的课程及其地点、日期和时间，其 SQL 语句为：

```
SELECT ClassName, Location, DayandTimes FROM Classes
```

我们创建一个 Statement 对象，并在 executeQuery()方法调用过程中传送 SQL 字符串，获取所需的这些数据。

```
//Declare a method and some variables.
public void ListClasses() throws SQLException {
    int i, NoOfColumns;
    String ClassName, ClassLocation, ClassSchedule;
    //Initialize and load the JDBC - ODBC driver.
    Class.forName ("jdbc.odbc.JdbcOdbcDriver");
    //Make the connection object.
    Connection Ex1Con = DriverManager.getConnection( "jdbc:odbc:
StudentDB;uid="admin";pw="sa");
    //Create a simple Statement object.
    Statement Ex1Stmt = Ex1Con.createStatement();
    //Make a SQL string, pass it to the DBMS, and execute the SQL statement.
    ResultSet Ex1rs = Ex1Stmt.executeQuery( "SELECT ClassName, Location,
DaysAndTimes FROM Classes");
    //Process each row until there are no more rows.
    // And display the results on the console.
    System.out.println("Class          Location          Schedule");
    while (Ex1rs.next()) {
        // Get the column values into Java variables
        ClassName = Ex1rs.getString(1);
        ClassLocation = Ex1rs.getString(2);
        ClassSchedule = Ex1rs.getString(3);
        System.out.println(ClassName, ClassLocation, ClassSchedule);
    }
}
```

正如所看到的，该程序非常简单。我们首先初始化连接并创建一个 Statement 对象。然后使用 executeQuery()方法调用传送 SQL 串，驱动程序将 SQL 字符串传送给 DBMS，DBMS 执行查询并返回结果。最后语句结束，丢弃优化的执行计划。

40.1.2　PreparedStatement

对于 PreparedStatement 对象，正如其名字所暗示的那样，应用程序使用 java.sql.Connection.preparedStatement()方法准备 SQL 语句。PreparedStatement()方法使用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

了一个 SQL 字符串,该字符串要被传送到底层的 DBMS。DDMS 经过语法分析、查询计划优化以及执行的生成等几个阶段,但并不执行 SQL 语句。DBMS 可能要返回一个被优化的执行计划句柄,该句柄由 JDBC 以内部方式存储在 PreparedStatement 对象中。

PreparedStatement 对象中的方法见表 40.2。注意,executeQuery( ) 、executeUpdate( )及 execute( )方法不使用任何参数。它们只是底层 DBMS 执行已经优化的 SQL 语句的调用。

表 40.2 PreparedStatement 对象的方法

返回类型	方法名称	参数
ResultSet	executeQuery	()
int	executeUpdate	()
boolean	execute	()

PreparedStatement 的主要特性之一是,它能处理 IN 类型的参数。这种参数是通过在 SQL 语句中使用"?"作为参数标记代替实际值的方法来说明的。在 Java 程序中,与参数的联系是由 setxxxx( )方法实现的,见表 40.3。所有的 setxxxx( )方法都使用参数索引,即 1 表示第一个"?",2 表示第二个"?"等等。

表 40.3 java.sql.PreparedStatement——与参数相关的方法

返回类型	方法名称	参数
void	clearParameters	()
void	setAsciiStream	(int parameterIndex, java.io.InputStream x,int length)
void	setBinaryStream	(int parameterIndex, java.io.InputStream x,int length)
void	setBoolean	(int parameterIndex,boolean x)
void	setByte	(int parameterIndex,byte x)
void	setBytes	(int parameterIndex,byte x[])
void	setDate	(int parameterIndex,java.sql.Date x)
void	setDouble	(int parameterIndex,double x)
void	setFloat	(int parameterIndex,float x)
void	setInt	(int parameterIndex,int x)
void	setLong	(int parameterIndex,long x)
void	setNull	(int parameterIndex,int sqlType)
void	setBignum	(int parameterIndex,Bignum x)
void	setShort	(int parameterIndex,short x)
void	setString	(int parameterIndex,String x)
void	setTime	(int parameterIndex,java.sql.Time x)
void	setTimestamp	(int parameterIndex,java.sql.Timestamp x)
void	setUnicodeStream	(int parameterIndex, java.io.InputStream x,int length)
高级特性——对象操作		
void	setObject	(int parameterIndex, Object x, int targetSqlType, int scale)
void	setObject	(int parameterIndex, Object x, int targetSqlType)
void	setObject	(int parameterIndex,Object x)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在使用 PreparedStatement 时，驱动程序实际上只向 DBMS 发送执行计划 ID 和参数。这样做的结果是减少了网络传输，非常适合于 Internet 上的 Java 应用程序。当一个 Java 应用程序需要多次执行 SQL 语句时，应该使用 PreparedStatement 对象。但是必须记住，即使在一个 Java 应用程序的执行过程中可以得到优化的执行计划，DBMS 也要在程序结束时丢弃执行计划。所以，在每次程序运行时，DBMS 必须经过创建一个执行计划所需的所有步骤。使用 PreparedStatement 对象可以获取比使用简单的 Statement 对象更快的 SQL 执行性能，因为 DBMS 无需对每个 SQL 语句的执行都运行一遍创建执行计划的所有步骤。

下面的程序实例给出了如何使用 PreparedStatement 类访问数据的例子。数据库的设计图见第 39 章“ JDBC : Java 数据库连接”。在该实例中，我们对在 Statement 实例中开发的例子进行优化。

参见 39.2.2“JDBC 应用程序剖析”。

程序清单 40.1 中的简单 Statement 实例可以使用几种主要方式进行优化。首先，DBMS 每次都建立一遍执行计划，改用 PreparedStatement。第二，查询列出了所有课程，可能会滚出窗口，可以通过建立带参数的查询改进这种情况，见程序清单 40.2。

```
//Declare class variables
Connection Con;
PreparedStatement PrepStmt;
boolean Initialized = false;
private void InitConnection() throws SQLException {
    //Initialize and load the JDBC - ODBC driver.
    Class.forName ("jdbc.odbc.JdbcOdbcDriver");
    //Make the connection object.
    Con = DriverManager.getConnection("jdbc:odbc:StudentDB;uid="admin";pw="sa");
    //Create a prepared Statement object.
    PrepStmt = Ex1Con.prepareStatement( "SELECT ClassName, Location, DaysAndTimes FROM Classes
WHERE ClassName = ?");
    Initialized = True;
}

public void ListOneClass(String ListClassName) throws SQLException {
    int i, NoOfColumns;
    String ClassName, ClassLocation, ClassSchedule;
    if (! Initialized) {
        InitConnection();
    }
    // Set the SQL parameter to the one passed into this method
    PrepStmt.setString(1, ListClassName);
    ResultSet Ex1rs = PrepStmt.executeQuery()
    //Process each row until there are no more rows and
    // display the results on the console.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        System.out.println("Class Location Schedule");
        while (Ex1rs.next()) {
            // Get the column values into Java variables
            ClassName = Ex1rs.getString(1);
            ClassLocation = Ex1rs.getString(2);
            ClassSchedule = Ex1rs.getString(3);
            System.out.println(ClassName,ClassLocation,ClassSchedule);
        }
    }
```

现在,如果一个学生想交互地检查一门课程的详细情况,可使用该样本程序。由于使用了PreparedStatement 对象,从第二次调用开始,我们就可以节省执行时间和网络传输量。

40.1.3 CallableStatement

对于一个安全、一致并且可管理的多层客户/服务器系统,应该允许数据访问使用预先存储的过程。预先存储的过程集中实现商务逻辑的可管理性并运行查询。运行于客户的 Java 小程序只有有限的资源,不能寄希望于这种小程序来运行巨型查询。但是查询的结果对于那些客户是很重要的。JDBC 允许通过 CallableStatement 类和转义子句字符串,使用预先存储的过程。

CallableStatement 对象是通过 Connection 对象中的 PrepareCall()方法创建的。PrepareCall()方法使用一个字符串作为其参数,该字符串称为转义子句(escape clause),其形式为:

{[? =] call <存储的过程名> [<参数>, <参数>...]}

CallableStatement 类支持使用参数,这些参数可以是来自预先存储过程的 OUT 类型参数,也可以是传送到预先存储过程的 IN 类型参数。参数的标记(问号)用于返回值(如果有的话)和任何输出参数,因为参数标记绑定于预先存储过程的程序变量。输入参数可以是常量或变元参数。对于动态参数化的语句,转义子句字符串的形式为:

{[? =] call <存储的过程名> [<? >,<? >...]}

在调用 executeQuery()、executeUpdate()或 execute() 方法之前,应当使用 registerOutParameter()方法(见表 40.4)对 OUT 参数进行登记。

表 40.4 CallableStatement——OUT 参数登记方法

返回类型	方法名称	参数
void	registerOutParameter	(int parameterIndex,int sqlType)
void	registerOutParameter	(int parameterIndex,int sqlType,int scale)

当预先存储过程执行完之后,DBMS 向 JDBC 驱动程序返回结果值。Java 程序使用表 40.5 中列出的方法可以访问该返回值。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 40.5　CallableStatement 参数访问方法

返回类型	方法名称	参数
boolean	getBoolean	(int parameterIndex)
byte	getByte	(int parameterIndex)
byte[]	getBytes	(int parameterIndex)
java.sql.Date	getDate	(int parameterIndex)
double	getDouble	(int parameterIndex)
float	getFloat	(int parameterIndex)
int	getInt	(int parameterIndex)
long	getLong	(int parameterIndex)
java.lang.Bignum	getBignum	(int parameterIndex, int scale)
Object	getObject	(int parameterIndex)
short	getShort	(int parameterIndex)
String	getString	(int parameterIndex)
java.sql.Time	getTime	(int parameterIndex)
java.sql.Timestamp	getTimestamp	(int parameterIndex)
其它功能		
boolean	wasNull	()

如果一个学生想要某一课程的成绩，数据库设计模型见第 39 章，那么我们需要在各种表上进行许多操作，如找出关于该学生的所有成绩、将其与课程名匹配、计算分数等等。这是很适合于预先存储过程的一个事务逻辑。在这一例子中，我们给预先存储过程一个学生 ID 和一个课程 ID，该过程返回一个所需的分数。客户程序非常简单，所有的处理都在服务器端进行。这就是需要使用 CallableStatement 语句的原因。

预先存储过程的调用形式为：

```
studentGrade = getStudentGrade(StudentID, ClassID)
```

在 JDBC 调用中，我们创建了一个 CallableStatement 对象并使用“?”符号作为其参数位置标识，然后将 Java 变量与参数连接起来，见程序清单 40.3。

```
public void DisplayGrade(String StudentID, String ClassID) throws SQLException {
    int Grade;
    //Initialize and load the JDBC - ODBC driver.
    Class.forName ("jdbc.odbc.JdbcOdbcDriver");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
//Make the connection object.
Connection Con = DriverManager.getConnection("jdbc:odbc:StudentDB;uid="admin";pw="sa");

//Create a Callable Statement object.
CallableStatement CStmt = Con.prepareCall("{? = call getStudentGrade[?,?]}");

// Now tie the placeholders with actual parameters.
// Register the return value from the stored procedure
// as an integer type so that the driver knows how to handle it.
// Note the type is defined in the java.sql.Types.
CStmt.registerOutParameter(1,java.sql.Types.INTEGER);

// Set the In parameters (which are inherited from the
// PreparedStatement class)
CStmt.setString(1,StudentID);
CStmt.setString(2,ClassID);

// Now we are ready to call the stored procedure
int RetVal = CStmt.executeUpdate();

// Get the OUT parameter from the registered parameter
// Note that we get the result from the CallableStatement object
Grade = CStmt.getInt(1);

// And display the results on the console.
System.out.println(" The Grade is : ");
System.out.println(Grade);
}
```

正如我们所看到的那样,JDBC 最小化了从预先存储过程获取结果的复杂程度。虽然它仍有些复杂,但已经比以前简单多了。也许将来的这些步骤会变得更为简单。

现在,我们已经看到了如何使用 SQL 与底层 DBMS 通信,下面我们来看看对于从数据库中作为 SQL 语句的结果所发送回来的结果进行处理还需要做些什么。

40.2 ResultSet 处理检查结果

ResultSet 对象实际上是一个管式数据集,即它是由以统一形式的列组织的数据行组成。在 JDBC 中,Java 程序一次只能看到一个数据行。程序使用 next() 方法走到下一数据行。JDBC 不提供任何沿着 ResultSet 向后移动或记忆行位置(在 ODBC 中称为书签)的方法。当程序获得一数据行后,它可以使用位置索引(第一列使用 1,第二列使用 2,等等)或列名称,以便使用 getxxxx()方法获得字段值。表 40.6 列出了与 ResultSet 对象有关的方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 40.6　java.sql.ResultSet 方法

返回类型	方法名称	参数
boolean	next	()
void	close	()
boolean	wasNull	()
通过列位置获得数据		
java.io.InputStream	getAsciiStream	(int columnIndex)
java.io.InputStream	getBinaryStream	(int columnIndex)
boolean	getBoolean	(int columnIndex)
byte	getByte	(int columnIndex)
byte[ ]	getBytes	(int columnIndex)
java.sql.Date	getDate	(int columnIndex)
double	getDouble	(int columnIndex)
float	getFloat	(int columnIndex)
int	getInt	(int columnIndex)
long	getLong	(int columnIndex)
java.lang.Bignum	getBignum	(int columnIndex, int scale)
Object	getObject	(int columnIndex)
short	getShort	(int columnIndex)
String	getString	(int columnIndex)
java.sql.Time	getTime	(int columnIndex)
java.sql.Timestamp	getTimestamp	(int columnIndex)
java.io.InputStream	getUnicodeStream	(int columnIndex)
通过列名称获得数据		
java.io.InputStream	getAsciiStream	(String columnName)
java.io.InputStream	getBinaryStream	(String columnName)
boolean	getBoolean	(String columnName)
byte	getByte	(String columnName)
byte[ ]	getBytes	(String columnName)
java.sql.Date	getDate	(String columnName)
double	getDouble	(String columnName)
float	getFloat	(String columnName)
int	getInt	(String columnName)
long	getLong	(String columnName)
java.lang.Bignum	getBignum	(String columnName, int scale)
Object	getObject	(String columnName)
short	getShort	(String columnName)
String	getString	(String columnName)
java.sql.Time	getTime	(String columnName)
java.sql.Timestamp	getTimestamp	(String columnName)
java.io.InputStream	getUnicodeStream	(String columnName)
int	findColumn	(String columnName)
SQLWarning	getWarnings	()
void	clearWarnings	()
String	getCursorName	()
ResultSetMetaData	getMetaData	()

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

可以看出,ResultSet 的方法虽然很多,但都非常简单。其中一个主要的方法是 getxxxx()。GetMetaData()方法返回关于 ResultSet 的元数据信息。DatabaseMetaData 以 ResultSet 的形式返回该结果。ResultSet 还包括使用于隐式 SQLWarnings 的方法。检查警告信息时使用 getWarning()方法是一个好办法,该方法在没有警告时返回 null。

40.3 其它 JDBC 类

现在,我们已经看到了所有与数据库有关的主要类,下面来看看 JDBC 中所提供的其它一些支持类。这些类包括 Date、Time、TimeStamp 等等。这些类中的大部分类扩展了基本的 Java 类,增加了对 SQL 来说为特殊数据类型的处理和转换功能。

40.3.1 java.sql.Date

该包(见表 40.7)为 Java 程序提供了处理使用年、月、日值的 SQL 日期信息的能力。

表 40.7 java.sql.date 方法

返回类型	方法名称	参数
Date	Date	(int year,int month,int day)
Date	Date	(long date)
Date	valueOf	(String s)
String	toString	()
int	getHours	()
int	getMinutes	()
int	getSeconds	()
void	setHours	(int Hr)
void	setMinutes	(int Min)
void	setSeconds	(int Sec)
void	setTime	(long date)

40.3.2 java.sql.Time

正如在表 40.8 中所看到的,java.sql.Time 把 Time 对象添加到 java.util.Date 包中,用来处理小时、分钟和秒。java.sql.Time 也用来表示 SQL 的 Time 信息。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 40.8　java.sql.Time 方法

返回类型	方法名称	参数
Time	Time	(int hour,int minute,int second)
Time	Time	(long time)
Time	Time	valueOf(String s)
String	toString	()
int	getDate	()
int	getDay	()
int	getMonth	()
int	getYear	()
void	setDate	(int date)
void	setMonth	(int month)
void	setTime	(int time)
void	setYear	(int year)

40.3.3　java.sql.Timestamp

java.sql.Timestamp 包将 TimeStamp 类添加到 java.util.Date 包中(见表 40.9)。它增加了毫微秒的处理能力。但是秒以下的时间戳的精度取决于数据库的字段及操作系统。

表 40.9　java.sql.Timestamp 方法

返回类型	方法名称	参数
TimeStamp	TimeStamp	(int year, int month, int date, int hour, int minute, int second, int nano)
TimeStamp	TimeStamp	(long time)
TimeStamp	valueOf	(String s)
String	toString	()
int	getNanos	()
void	setNanos	(int n)
boolean	after	(TimeStamp ts)
boolean	before	(TimeStamp ts)
boolean	equals	(TimeStamp ts)

40.3.4　java.sql.Types

java.sql.Types 类定义了一个用来标识 SQL 类型的等价于整型常量的 XOPEN 集。这些常量为 final 类型。所以,在应用程序和小程序中不能对其进行重新定义。表 40.10 列出了这些常量的名字及其值。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 40.10 java.sql.Types 常量

常量名	值	常量名	值
BIGINT	-5	NULL	0
BINARY	-2	NUMERIC	2
BIT	-7	OTHER	1111
CHAR	1	REAL	7
DATE	91	SMALLINT	5
DECIMAL	3	TIME	92
DOUBLE	8	TIMESTAMP	93
FLOAT	6	TINYINT	-6
INTEGER	4	VARBINARY	-3
LONGVARBINARY	-4	VARCHAR	12
LONGVARCHAR	-1		

40.3.5 java.sql.DataTruncation

java.sql.DataTruncation 类提供了当一个 SQL 语句产生 DataTruncation 警告或异常时获取其详细信息的方法。数据截取可能发生于列值或参数。

DataTruncation 对象的主要元素有：

- 索引给出列或参数号。
- 如果截取发生在参数则 parameter 标志为 true(真)，如果截取发生在列则 parameter 标志为 false(假)。
- 如果截取发生于读取数据的过程中则 read 标志为 true(真)，如果截取发生于写入数据的过程中则 read 标志为 false(假)。

DataTruncation 对象还包括一个数据大小元素，该元素具有截取的值的实际大小(以字节为单位)和转换大小，转换大小指的是实际转换的字节个数。

如表 40.11 所示，有各种方法可以使 Java 程序检索这些元素的值。例如，如果数据截取发生于读取数据时，则 getRead()方法返回 true(真)值，如果发生于写数据期间，则返回 false(假)值。

表 40.11 java.sql.DataTruncation 方法

返回类型	方法名称	参数
int	getDataSize	()
int	getIndex	()
boolean	getParameter	()
boolean	getRead	()
int	getTransferSize	()

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

40.4　JDBC 的前景

在 Java 语言提升到 Java 平台这一正确走向中，JDBC 是重要一步。Java API——包括企业版 API(JDBC、RMI、系列化及 IDL)、安全性 API 及服务器 API ——是开发企业级、分布式、多层客户/服务器应用程序的基本组成部分。

JDBC 规范的生命周期以 Net 的速度进行循环——每个 Net 年计时要等于 7 个普通年。JDBC 规范是固定的。所以开发人员和驱动程序供应商并不是在追求一个移动的目标。

JDBC Compliant

Sun 公司已经为驱动程序设立了 JDBC Compliant 规范。如果它通过了由 Sun 公司和 Intersolv 公司开发的 JDBC 符合性测试，则一个特殊的驱动程序将被称之为 JDBC Compliant。目前，一个驱动程序要通过符合性测试应至少支持 ANSI SQL 92 Entry Level。

JDBC Compliant 规范对于开发人员非常有用，因为使用 JDBC Compliant 驱动程序开发人员可以充满信心地使用 JDBC 开发应用程序，并且可以确保对数据库的访问(在客户端)。

JDBC 之所以倍受欢迎的另一个因素是它与 ODBC 的相似性。Sun 公司做出了正确的选择，即遵守 ODBC 的思想和抽象方法。这样就使得 ISV 和用户很容易将他们在 ODBC 的经验及现有的 ODBC 驱动程序转移使用到 JDBC。在 JDBC 规范中，这一目标被描述为“JDBC 必须在公共数据库接口的顶层实现”。

通过使 JDBC 成为 Java 语言的一部分，我们可以充分发挥出 Java 语言对于数据库访问的优势。同样，由于所有的实现必须支持 Java API，JDBC 已经变成了一个通用的标准。这一思想在 JDBC 规范中描述为“提供一个由 Java 系统的其余部分组成的 Java 接口”，这使得 JDBC 成为了基于 Java 进行数据库开发的理想候选产品。

另一个好的设计思想是驱动程序独立于 JDBC。底层数据库驱动程序既可以是一个本地库——如用于 Windows 系统的 DLL，也可以是连接于侦听器的 Java 过程。JDBC 完整的 Java 实现适用于各种网络和其它 Java OS 计算机，这样就使得 JDBC 成为了一个多功能 API 的集合。

> **注意**：依笔者的拙见，JDBC 最重要的优点是其简单性和多功能性。JDBC 设计者的最终目标是保持 API 和普通情况的简单化，并“在各种独立接口上支持奇异的东西”。另外，他们要求对多重功能使用多个方法。即使在第一版本中，他们也已经达到了自己的目标。
>
> 例如，Statement 对象对于返回数据行的 SQL 语句有 executeQuery()方法，对于没有数据返回的语句有 executeUpdate()方法，而对于返回多个 ResultSet 的这种特殊情况，有一个单独的方法 execute()。

随着使用 JDBC 所开发出的应用程序的增多，以及 Java 平台的更加成熟，必将会有越来越

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

多的特性增加到 JDBC 中。我们所需要的一个特性是希望它具有更加多样性的数据光标，特别是对于客户/服务器的处理。目前的设计将数据光标的管理细节留给了驱动程序。对于可滚动数据光标、定位更新/删除功能等，笔者更喜欢应用层的控制。另一个相关的特性是书签特性，这一特性特别适用于分布式处理环境如 Internet 环境。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第六部分　基于组件的开发

第 41 章　JavaBeans
第 42 章　JavaIDL：Java 的 CORBA 接口
第 43 章　Java – COM 集成

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 41 章　JavaBeans

本章内容：

41.1　自包含的组件
41.2　组件模型中的重要概念
41.3　设计 JavaBean 的基础
41.4　创建和使用属性
41.5　使用事件与其它组件通信
41.6　内省：建立和使用 BeanInfo 类
41.7　定制：提供定制属性编辑器和 GUI 接口
41.8　企业级 JavaBeans

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

41.1 自包含的组件

JavaBeans 向 Java 平台增加了通过简单地将自包含组件连接在一起,创建一个完整应用程序的功能。微软的 Visual Basic 和 Borland 的 Delphi,都是这种应用软件的例子,它允许用户通过结合相互独立的组件建立完整的应用程序。这两个应用软件的成功和流行本身就告诉我们这种建立应用程序的风格已取得了巨大成功。

就像其它的模型一样,JavaBeans 组件在大小或复杂程度上也不存在任何限制。原则上讲,JavaBeans 组件(或简称为 Beans)可以是从小的器件和控件到大的容器和应用程序等各种组件。JavaBeans 的思想是,对前者提供易实现的功能,对后者提供充分的灵活性。在这种思想的鼓舞下,读者将看到创建和使用 Beans 是多么地简单。但是,当读者阅读完本章后,如果需要的话将会学习到足以创建更大更复杂 Beans 的本领。如果读者需要更深入地钻研 JavaBeans 复杂的细节,或许需要购买 Que 公司的《Special Edition Using JavaBeans》一书,该书更详细地介绍了本章所讨论的所有课题。

41.2 组件模型中的重要概念

JavaBeans 提供了一个与平台无关的组件体系结构。非平台无关组件结构的例子有 Windows 平台的 COM/OLE 和 Macintosh 平台的 OpenDoc。一个为 OpenDoc 容器所编写的组件,例如 ClarisWorks,不能使用于 COM/OLE 容器如 Microsoft Word。由于 JavaBeans 独立于体系结构,Beans 可以放置到任何容器中,只要 JavaBeans 与该容器的组件体系结构之间存在一个桥。所以,JavaBeans 组件可使用于 Microsoft Word 和 ClarisWorks 中的任何一个。为实现这一看起来似乎并不重要的事情,JavaBeans 规范采用了其它流行组件模型普遍具有的特性。特别是以下这些特性:

- 组件字段或属性
- 组件方法或函数
- 事件及相互通信
- 状态保持和存储

> **注意**:读者如果已经熟悉组件模型的概念,那么不必再阅读本节的内容,可以直接跳到下一节"设计 JavaBeans 的基础"

41.2.1 组件字段或属性

对一个有用的组件来说,必须有一组属性用来定义其状态。例如,如果我们要设计用来显示某些文本的一个组件,该组件的属性之一可能是字符的前景色,另一个属性可能是字符的大小和字体。作为一个整体来看,这样的一组描述组件的属性也定义了它的状态。例如,如果一个组件的属性完全匹配于另一个组件,则二者处于同样的状态。

属性通常不仅用于定义组件的外形,而且还用于定义组件的行为。这是因为有的组件不需要有任何外形。例如,电子数据表中的一个组件可能用于计算一些资金列的利息。如果这

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

个组件不用来显示数据，则它可能不应该有任何与外形有关的属性。然而，该组件可能将具有一个定义当前利率的属性。

属性可以是布尔值、字符串、数组及其它组件，它们可以相互依赖。对于上述例子，显示金额列的组件可能需要得到通知，如果其它组件的利率属性改变的话。

41.2.2　组件方法或函数

可以说，一个组件的 API 是一组方法或函数，这些方法或函数可以被其它组件和容器所调用。对一个容器来说必须有办法修改组件的属性，留意事件的发生（见稍后的内容），或执行一些功能。

不同的组件模型在如何创建供其它组件使用的组件属性和方法时是有差别的。因为关于不同模型的实现方法已经有完整的书籍编写出来，因此可以说这是组件模型的普遍特性。作为与 JavaBeans 的关系，这一话题将在本章稍后的内省（Introspection）一节中讨论。

41.2.3　事件和相互通信

一个组件就其本身而言只是一个孤立的组件。即使有些组件有扩展的功能和诸多的属性。就组件的实质来说，只有当与其它组件结合时才是有用的。所以如果在一个容器中两个组件放置在一起，它们是如何交流的呢？一个组件是如何让另外一个组件知道它已经做完了某件事，而这些也正是另一组件所应该知道的呢？

大多数组件通信的方法是使用事件传输（event transmission）。一个组件（或一个容器）经历一些动作便引起产生一个事件。例如，当我们单击鼠标按钮时便产生一个事件。按照所使用的模型，该组件将通知容器、相关的组件或二者关于事件的信息。同时，环境中的对象将根据传播的事件进行动作。例如，当文件（File）对话框侦听到我们单击 Browse 按钮时，将显示出自身的内容。

41.2.4　状态维持和存储

对组件来说记住其状态是很重要的。这一点太普通了，我们或许并没有认识到它。当我们打开一个应用程序时，它能记起其上次关闭时在窗口中的大小和位置，程序在保持（某种程度上）持续的状态。

存储和检索组件的能力也很重要。Sun Microsystems 公司称之为打包（packaging）。这对于可能在网络上的分布式环境中服务的组件尤为重要。

41.3　设计 JavaBean 的基础

所有优秀的程序员都知道设计阶段在编程中的重要性。所以，我们将从讲解如何设计一个 Bean 开始。随着以后的学习，设计 Bean 的方法将会直接影响到 Bean 在容器中的行为。例如，为方法选择名字时应遵循指定的设计规范。如果一开始就牢记这些规则，要使自己的 Bean 加入到内省中，作为编程者，不需要再进行附加的编程。我们现在不必为内省究竟是什么而焦虑，稍后就会讲到。

Bean 的设计包括以下几步：

1. 指定 Bean 的属性

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

2. 指定 Bean 所产生或响应的事件

3. 定义 Bean 应公开给其它 Bean 或容器的属性、方法及事件

4. 确定复杂的问题，例如 Bean 是否有其自己的定制对话框，或者是否需要一些典型的信息

我们将从具体设计一些 Bean 入手。对于所使用的这个简单例子，假设要开发两个 Bean，其中一个 Bean 允许文本输入到其中，另一个则显示一些文本。我们可以设想一下这些 Bean 的使用情况，通过将这两个 Bean 放于一个容器内，我们可以使用其中的一个来输入文本，而另一个显示这些文本。读者可以想一想这些 Bean 需要哪些类型的属性呢？这些 Bean 需要侦听的事件有哪些？这些 Bean 将产生哪些事件呢？这些 Bean 将公开其所有的属性和事件，还是只公开其中的一部分呢？在这一点上，我们可能并不知道这些问题的答案。此处，过程是一个最为重要的概念，随着本章的进展所有这些问题的细节将会逐渐明了。然而，任何一个 Bean 所需要的第一件东西就是它的名字，本章中样本 Bean 的名字分别为 TextDisplayer 和 TextReader。

41.3.1 指定 Bean 的属性

TextDisplayer 和 TextReader Bean 肯定需要一个用来定义它所拥有的文本的属性。例如，对 TextDisplayer Bean 还包括定义背景色、字体、以及字符颜色的属性。而对 TextReader Bean 还包括定义它所能显示的字符列数的属性。表 41.1 列出了 TextDisplayer Bean 的属性及实现它们时所使用的 Java 类型。表 41.2 列出了 TextReader Bean 的属性及其所使用的 Java 类型。

表 41.1 TextDisplayer Bean 的属性及 Java 类型

属性名	Java 类型
OutputText	java.lang.String
BGColor	java.awt.Color
TextFont	java.awt.Font
FontColor	java.awt.Color

表 41.2 TextReader Bean 的属性及 Java 类型

属性名	Java 类型
InputText	java.lang.String
Width	int

41.3.2 指定 Bean 所产生或响应的事件

TextDisplayer Bean 必须响应的一个事件是指定其文本修改。特别是，它必须更新其 OutputText 属性并重新绘制其自己。TextReader Bean 不需要响应任何事件，但当用户修改其文本属性时它必须产生(或击活)一个事件，这种类型的事件显然应称之为属性改变事件(PropertyChangeEvent)。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

41.3.3　属性、方法及事件的公开

由于这些 Bean 非常简单，所以对于 Bean 的容器和其它对它们感兴趣的 Bean 不需要隐藏任何东西。JavaBean 提供了一个机制，这种机制将使用方法的名字来展开属性和事件的名字和类型。可以肯定随着后面的学习，将会知道这一机制究竟是如何工作的。在本章稍后，将介绍如何明确定义一个 Bean 中的哪些属性、方法及事件向其环境公开。

41.3.4　初始化属性值和 Bean 的定制

我们需要使这个例子尽量地保持简单，所以假设 Bean 不需要任何原型信息（我们将对所有属性定义缺省值），并且它们也没有其自己的定制对话框。这意味着这些 Bean 在实例化时有一个预定义的状态，并且其属性使用标准的 PropertyEditors 属性编辑器。例如，如果正在设计一个显示 HTML 页的 Bean，有可能无法指定缺省值。在 Bean 实例化时我们需要知道所显示的文件。表 41.3 显示了 TextDisplayer Bean 的属性及其所使用的缺省值。同样，表 41.4 是 TextReader Bean 的属性及其使用的缺省值。

表 41.3　TextDisplayer Bean 的缺省属性值

属性名	缺省值
TextOutput	"TextDisplayer"
BGColor	java.awt.Color.white
TextFont	Courier,normal,12
FontColor	java.awt.Color.black

表 41.4　TextReader Bean 的缺省属性值

属性名	缺省值
TextInput	""（一个空串）
Width	40

到此，我们的 Bean 设计可以开始进行编码了。由于我们尚未学习如何使一个 Bean 完成某些操作的内容，这将是一个渐进的过程。所有实际显示这一 Bean 所需要的代码不包括在程序清单 41.1 和 41.2 中，因为这主要与 AWT 有关，不属于本章的内容。如果要查看完整的程序清单，请参考 CD－ROM。从图 41.1 可以看到所建立的 Bean 在 BeanBox 中的工作情况。BeanBox 是一个 JavaBean 容器，可以从 Sun 公司的 Web 站点上卸载下来了，它包括在 BDK (Beans Development Kit)中。到现在为止，所需要的 Bean 已经完全出来了。用于实例化 TextDisplayer Bean 的源代码在程序清单 41.1 中给出。



图 41.1　显示 TextDisplayer 和 TextReader Bean 的 Sun 公司的 BeanBox

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public class TextDisplayer extends Canvas implements PropertyChangeListener {
    // default constructor for this Bean. This is the constructor that an
    // application builder (like Visual Basic) would use.
    public TextDisplayer() {
        this("TextDisplayer", Color.white, new Font( "Courier", Font.PLAIN, 12 ),
            Color.black );
    }

    // custom constructor for this Bean. This is the constructor you would
    // likely use if you were going to do all your coding from scratch.
    public TextDisplayer( String OutputText, Color BGColor, Font TextFont,
                          Color FontColor ) {
        super(); // call the Canvas's constructor.
        this.OutputText = OutputText;
        this.BGColor = BGColor;
        this.TextFont = TextFont;
        this.FontColor = FontColor;

        setFont( TextFont );            // set the Canvas's font.
        setBackground( BGColor );       // set the Canvas's background color.
        setForeground( FontColor );     // set the Canvas's foreground color.
    }

    // this Bean's properties.
    protected String OutputText;
    protected Color BGColor, FontColor;
    protected Font TextFont;
}
```

可以看出，我们指定 Bean 实现了一个称之为 PropertyChangeListener 的接口。这是为了 TextDisplayer 能够在收到一个事件时更新其 TextOutput 属性，其工作方式将在本章稍后进一步详细讨论。实例化 TextReader Bean 所需要的源代码在程序清单 41.2 中给出。

程序清单 41.2 TextReader.java——实例化 TextReader Bean 的源代码

```
public class TextReader extends TextField {
    // default constructor for this Bean. This is the constructor that an
    // application builder (like Visual Basic) would use.
    public TextReader() {
        this("",40);
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    // custom constructor for this Bean. This is the constructor that you would
    // likely use if you were doing your coding from scratch.
    public TextReader( String InputText, int Width ) {
      super( InputText, Width );
      this.InputText = InputText;
      this.Width = Width;
      setEditable( true );
    }

    // this Bean's properties.
    protected String InputText;
    protected int Width;
}
```

41.4　创建和使用属性

在图 41.1 中可以看出，TextDisplayer Bean 显示其本身时使用了白色的背景色和黑色的文本色，这是由其属性设置决定的。如果将 FontColor 属性设置为 red，那么它将使用红色显示文本。如果一个组件的属性不能被其它 Bean 改变，则该组件的使用性和再使用性将降低。例如，如果在一个帐目数据包内使用 TextDisplayer Bean，则需要改变 Bean 的 FnotColor 属性为 red，用来表示一个负数值。所以，如何让其它 Bean 知道它们可以设置（或读取）该属性呢？如果正在从草稿开始进行编码，则可以查看关于 Bean 的文档资料，但是如果是使用一个应用程序构造器时又如何实现呢？幸运的是，有一种方法可以实现而无需增加任何额外的代码，稍后我们将看到这一点。

JavaBean 支持两种类型的属性：单值属性和索引属性。另外，属性也有绑定和限制性属性之说。单值属性是只有一个值的属性。顾名思义，索引属性有多个值，且每个值对应一个唯一的索引。如果一个属性是绑定性属性，指的是其它一些 Bean 依赖于该属性。在上面的例子中 TextReader Bean 的 InputText 属性是一个绑定于 TextDisplayer Bean 的属性，当其 InputText 字段修改之后，TextReader 必须通知 TextDisplayer。如果一个属性在改变之前必须检查其它组件，则这个属性为限制性属性。注意，限制性属性不能强制改变——可能有一个或多个组件不允许修改该值。

41.4.1　单值属性

所有属性值的访问都是通过调用其所属 Bean 对象的方法实现的。可读属性有一个 getter 方法，用于读取其属性值。可写属性有一个 setter 方法，用于改变属性的值。这些方法并非限制在只是简单地返回属性的值，它们还可以执行一些运算，返回一些其它的值。我们所建立的 Bean 的所有属性都是单值属性。

到此为止，我们可以开始介绍内省了。其它组件了解我们所建立 Bean 的属性的方法要依赖于某些条件。一般来讲，这一过程称为内省。事实上，java.beans.Introspector 类就是为其它

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

组件提供这些信息的类。Introspetcor 类遍历一个特殊 Bean 的类分层结构，如果发现 Bean 所提供的显式信息就使用它。但是，对那些不提供信息的 Bean，它使用设计模式隐式地从中析取信息。注意，这正是对我们的 Bean 所发生的事情。在定义访问方法使得 Introspector 类可以完成其任务时，使用了特殊的设计规则。如果选择使用其它名字，仍然可以公开一个 Bean 的属性，但是这需要提供一个 BeanInfo 类。关于 BeanInfo 类的详细介绍，参阅关于内省(Introspetion)一节。下面是一个可以使用的设计模式：

```
public void set < PropertyName > ( < PropertyType > value);
public < PropertyType > get < PropertyName > ();
public boolean is < PropertyName > ();
```

注意，最后的模式对于 Boolean 属性只是一个可选择的 getter 方法。setter 方法在程序清单 41.3 中给出。注意，所有的访问方法都声明为 synchronized。即使在这个 Bean 中没有严重的事件发生，也应该假设我们的 Bean 总是运行于一个多线程环境下的。使用同步的(synchronized)访问方法可以避免形成竞争的局面。读者可以检查 CD－ROM 的 TextReader.java 文件，查看 TextReader Bean 的访问方法。

```
public synchronized String getOutputText() {
    return( OutputText );
}

public synchronized void setOutputText( String text ) {
    OutputText = text;
    resizeCanvas();
}

public synchronized Color getBGColor() {
    return( BGColor );
}

public synchronized void setBGColor( Color color ) {
    BGColor = color;
    setBackground( BGColor );      // set the Canvas's background color.
    repaint();
}

public synchronized Font getTextFont() {
    return( TextFont );
}

public synchronized void setTextFont( Font font ) {
    TextFont = font;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        setFont( TextFont );            // set the Canvas's font.
        resizeCanvas();
    }

    public synchronized Color getFontColor() {
        return( FontColor );
    }

    public synchronized void setFontColor( Color color ) {
        FontColor = color;
        setForeground( FontColor ); // set the Canvas's foreground color.
        repaint();
    }
```

图 41.2 给出了 Sun 公司的 BeanBox 为 TextDisplayer Bean 所显示的属性单。注意，我们也可以看到其父类的属性。此处的 Bean 是 java.awt.Canvas 的继承，java.awt.Canvas 是 java.awt.Component 的继承，而 java.awt.Component 是 java.lang.Object 的继承。我们所看到的附加属性来自于 java.awt.Component 类。这说明了使用自动 JavaBean 内省方法的回退原理。在我们自己的 Bean 中，这可能是提供 BeanInfo 类的动机。同样，更详细的介绍见内省一节。

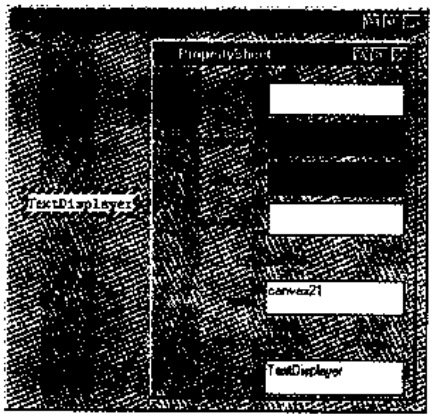


图 41.2　Sun 公司的 BeanBox 显示 Bean 的公开属性的属性单。注意父类的属性

41.4.2　索引属性

所有的索引属性必须是 Java 整型数。属性可以单个读取，也可以使用一个数组整体读取。索引属性的设计模式如下所示：

```
public < PropertyType > get< PropertyName > (int index);
public void set< PropertyName > (int index, < PropertyType > value);
public < PropertyType >[] get< PropertyName > ();
public void set< PropertyName > ( < PropertyType >[] value);
```

为了说明这一点，假设有一个 Meal 属性，该属性由一个 Courses 数组组成：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public Course getMeal (int course);
public void setMeal (int course,Course dish);
public Course[ ] getMeal ( );
public void setMeal (Course[ ] dishes);
```

41.4.3 绑定性属性

作为程序员,我们可以决定 Bean 的哪些属性可供其它组件绑定。为了提供自己的 Bean 中的绑定性属性,必须定义如下的方法:

```
public void addPropertyChangeListener(PropertyChangeListener l);
public void removePropertyChangeListener(PropertyChangeListener l);
```

为了对单一属性提供这种功能,应使用如下的设计模式:

```
public void add< PropertyName > Listener(PropertyChangeListener l);
public void remove< PropertyName > Listener(PropertyChangeListener l);
```

对于需要绑定到其它组件属性的 Bean,应实现 PropertyChangeListener 接口,它由下述方法组成:

```
public void propertyChange(PropertyChangeEvent evt);
```

不管什么时候,当一个 Bean 的绑定性属性更新后,所有与之一起注册的组件都必须调用 PropertyChange()方法。Java.beans.PropertyChangeSupport 类就是用来帮助完成这一过程的。程序清单 41.4 给出了在 TextReader Bean 中允许 InputText 属性被绑定所需的代码。

```
// setter method for the InputText property.
public synchronized void setInputText( String newText ) {
  String oldText = InputText;
  InputText = newText;
  setText( InputText );
  changeAgent.firePropertyChange("inputText", new String( oldText ),new String( newText ) );
}

// these two methods allow this Bean to have bound properties.
public void addPropertyChangeListener( PropertyChangeListener l ) {
  changeAgent.addPropertyChangeListener( l );
}

public void removePropertyChangeListener( PropertyChangeListener l ) {
  changeAgent.removePropertyChangeListener( l );
}

protected PropertyChangeSupport changeAgent = new PropertyChangeSupport( this );
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

41.4.4　限制性属性

在自己的代码中提供限制性属性的过程也非常简单，我们必须在 Bean 中定义下述方法：

```
public void addVetoableChangeListener(VetoableChangeListener l);
public void removeVetoableChangeListener(VetoableChangeListener l);
```

正如绑定性属性一样，使用下述设计模式可使单一属性成为限制性属性：

```
public void add<PropertyName>Listener(VetoableChangeListener l);
public void remove<PropertyName>Listener(VetoableChangeListener l);
```

用于限制其它组件属性的 Bean 应实现 VetoableChangeListener 接口，它包括如下方法：

```
public void vetoableChange(PropertyChangeEvent evt);
```

不管什么时候，当一个 Bean 中的限制性属性更新后，所有与其一起注册的组件都必须调用 vetoableChange( ) 方法。对此也有一个支持类用来帮助实现这一过程，使用 java.beans.VetoableChangeSupport 类，可帮助管理自己的 Vetoable 属性。程序清单 41.5 给出了使 TextReader Bean 的 Width 属性成为限制性属性所需的代码。

```
// setter method for the Columns property.
    public synchronized void setWidth( int newWidth )
    throws PropertyVetoException {
      int oldWidth = Width;
      vetoAgent.fireVetoableChange("width", new Integer( oldWidth ),new Integer( newWidth ) );
      // no one vetoed, so change the property.
      Width = newWidth;
      setColumns( Width );
      Component p = getParent();
      if ( p != null ) {
        p.invalidate();
        p.layout();
      }
      changeAgent.firePropertyChange("width", new Integer( oldWidth ),new Integer( newWidth ) );
    }

    // these two methods allow this Bean to have constrained properties.
    public void addVetoableChangeListener( VetoableChangeListener l ) {
      vetoAgent.addVetoableChangeListener( l );
    }

    public void removeVetoableChangeListener( VetoableChangeListener l ) {
      vetoAgent.removeVetoableChangeListener( l );
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
protected VetoableChangeSupport vetoAgent = new VetoableChangeSupport( this );
```

在这一特殊的示例中,我们选择 Width 属性既是绑定性的,又是限制性的。属性并非必须既是限制性的又是绑定性的。例如,为了使 Width 属性为限制性属性,而非绑定性属性,应从程序清单 41.5 中删除下述代码行:

```
changeAgent.firePropertyChange("Width",new Integer(oldWidth),
    new Integer(newWidth));
```

41.5 使用事件与其它组件通信

JavaBeans 组件模型的总体思想是提供一种创建可再用组件的方法。为了实现这一目的,Beans 在其环境中必须能够与其它 Beans 以及它的容器进行通信。这是通过 Listener(侦听)接口来完成的。从上一节中的 PropertyChangedEvent 中我们已经看到了有关这一方面的一些内容,下面详细介绍其运作方式。

Beans 使用与 AWT 相同的事件处理机制。也就是说,如果我们的 Beans 需要侦听来自于其它 Bean 的事件,它必须与那个 Bean 一起注册。为了做到这一点,必须对所感兴趣的事件实现 Listener 接口。同时,如果我们的 Bean 不再有兴趣侦听其它 Bean 的事件,则必须退出与那个 Bean 的注册。Bean 所要击发的任何事件必须从 java.util.EventObject 类继承而来。对于简单事件,java.util.EventObject 类本身就可以使用;然而,就像使用 java.lang.Exception 一样,使用子类会更加清晰,所以人们更倾向于使用子类。所有的 Listener 接口必须是 java.util.EventListener 接口的继承,此处同样也是使用子类更加方便。Listener 接口的事件处理方法应遵循内省(Introspection)设计模式,如下所示:

```
void < EventOccuranceName > (EventObjectType evt);
```

注意,< EventObjectType > 必须是 java.util.EventObject 的继承。下面是对于 DinnerServedListener 接口的一个事件处理器示例:

```
void dinnerServed(DinnerServedEvent evt);//DinnerServedEvent 从
                                         //java.util.EventObject 继承而来
```

事件处理器方法抛出异常没有任何限制。任何一个 Listener 接口也可以有任意多个相关的事件处理器。

组件可以侦听到的事件有两种类型:多点发送事件和单点发送事件。

41.5.1 多点发送事件

多点发送事件是最常用的文件类型。已经公开的 PropertyChangeEvent 是一个多点送事件,这是因为它可以有任意多个侦听者。在上面的例子中,我们使用了 addPropertyChangeListener()和 removePropertyChangeListener()方法,允许其它组件与这个 Bean 一起注册,以侦听一个绑定性属性的更新。对其它种类的多点发送事件该过程是一样的,注册方法应遵循内省(Introspection)设计模式,如下所示:

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
public synchronized void add< ListenerType( < ListenerType > listener);
public synchronized void remove< ListenerType > ( < ListenerType > listener);
```

关键字 synchronized 实际上并不是该设计模式所要求的内容。此处使用它是针对出现竞争这种情况的，而这种情况是可能发生的，特别是对于事件模型，必须采取预防措施。

41.5.2　单点发送事件

单点发送事件不像多点发送事件那样普遍，但单点发送事件也非常有用。单点发送事件只能有一个侦听者，如果更多的其它组件试图侦听一个单点发送事件，将产生一个 java.util.TooManyListenersException 异常。声明单点发送事件时应使用如下的设计模式：

```
public synchronized void add< ListenerType > ( < ListenerType > listener)
    throws java.util.TooManyListenersException;
public synchronized void remove< ListenerType > ( < ListenerType > listener);
```

41.5.3　事件适配器

有些情况下，有必要建立一个事件适配器，用以传送一个事件到一个组件。尤其是对于应用程序构造器更为实用，因为应用程序在运行之前并不知道组件之间如何连接以及如何使用相互的事件进行交互的。

事件适配器通过为另一个组件解释事件的一般含义，干预一般事件的处理方案。例如，假设用户在一个应用程序构造器中放置了一个按钮和一个文本框，如果要求当按了按钮时文本框内填充“Pressed”文本，则应用程序构造器可以使用一个事件适配器调用一个方法，该方法包含实现这一操作所需的由用户所产生的代码。如下是其最终的运作方式：

1. 事件适配器注册于事件源。换句话说，它在事件源组件中调用一个 addSomeEventListener()方法。

2. 事件源组件通过调用该事件适配器的事件处理器方法 someEvent()，击发一个事件。必须记住，事件源组件并不关心它是否正在调用一事件适配器。在这一点上，随着事件的击发，它可以继续其自己的事务。

3. 事件适配器在最终目标组件调用指定的由用户设计的方法。

4. 用户所设计的方法中的代码使用“Pressed”文本填充文本框组件。

有些情况下查看一些代码是有帮助的。程序清单 41.6 包含了一些伪代码，读者可以检查一下，看看事件适配器是如何编写的。示例中的代码是按照前面所给出的过程建立的，我们不能对这些代码进行编译（注意类关键字都改成了伪类 pseudoclass），但仍可以将其作为建立 Bean 的例子。



```
// this pseudoclass example uses a unicast mechanism to keep things simple.

public interface SomeEventListener extends java.util.EventListener {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public someEvent( java.util.EventObject e );
}

public pseudoclass button extends java.awt.Button {
    public void synchronized addSomeEventListener( SomeEventListener l )
                                        throws java.util.TooManyListenersException {
    if ( listener != null ) {
          listener = l;
      } else throw new java.util.TooManyListenersException;
    }

    private void fireSomeEvent() {
      listener.someEvent( new java.util.EventObject( this ) );
    }

    private SomeEventListener listener = null;
}

public pseudoclass eventAdaptor implements SomeEventListener {
    public eventAdaptor( TargetObject target ) {
      this.target = target;
    }

    someEvent( java.util.EventObject e ) {
      // transfer the event to the user generated method.
      target.userDefinedMethod();
    }

    private TargetObject target;
}

public pseudoclass TargetObject {
    public TargetObject() {
      adaptor = new eventAdaptor( this );
    }

    public userDefinedMethod() {
      // user generated code goes here.
    }

    private eventAdaptor adaptor;
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

41.6　内省:建立和使用 BeanInfo 类

在前面的几节和所设计的两个 Bean 例子中,读者已经看到了如何使用设计模式实现自动内省(Introspection)。读者也可能已经注意到自动内省机制是不完善的。如果回过头来看一下图 41.2,就可以看到这方面的情况。内省可能是 JavaBean 最重要的一个方面,因为没有内省,容器除了显示出来,不能再做任何事情。随着对设计 Bean 的进一步熟悉,我们将会发现有时候需要为 Bean 的最终用户提供附加的内省信息。对于自己的 Bean,也就是隐藏其父类的属性,以防止出现含混不清的情况。

本章前面已讨论过的 java.beans.Introspector 类,完成所有的模式分析工作,以公开一个组件所具有的属性、方法以及事件。作为第一步,这个类查看是否为正在考察的 Bean 已定义了 BeanInfo 类。如果找到,就不再在 BeanInfo 类提供信息的 Bean 区域内做任何模式分析工作。这意味着可以有选择地选择需要提供的信息和需要通过分析派生出的信息。为说明这一点需要为 TextDisplayer Bean 设计一个 BeanInfo 类。

现在要做的第一件事情是,定义需要提供的信息和将要留给 Introspector 类去分析的信息。对于我们的例子。表 41.5 给出了 TextDisplayer Bean 的属性名及所要显示的用户友好名,使用所定义的这些信息,就可以开始进行我们的 BeanInfo 类的工作了,即 TextDisplayerBeanInfo.class。注意"BeanInfo"是如何简单地添加到类名的。这就是内省(Introspection)的设计模式,Introspector 类通过添加"BeanInfo"到当前正在分析的 Bean 类名的方法,查找 BeanInfo 信息。

表 41.5　TextDisplayer Bean 的属性及其用户友好名

属性名	用户友好名
OutputText	"Text String"
BGColor	"Background Color"
TextFont	"Text Font"
FontColor	"Text Color"

所有的 BeanInfo 类必须实现 java.beans.BeanInfo 接口。初看起来这是很难的,java.bean.BeanInfo 接口中有八个方法。但是要知道,Introspector 类中有一个设置查看信息方式的过程。为清楚起见,我们对这一过程说明如下:

1. Introspector 类为其正在分析的 Bean 查找 BeanInfo 类。

2. 如果存在一个 BeanInfo 类,则调用 BeanInfo 类中的每个方法,查看它是否可以提供信息。Introspector 类将使用隐含分析法,为拒绝任何信息(返回一个 null 值)的 BeanInfo 类公开信息。如果没有找到 BeanInfo 类,Introspector 类将使用隐含分析法,分析 java.bean.BeanInfo 接口中的所有方法。

3. 然后,Introspector 类检查 BeanInfo 接口中的每个方法是否已经获得显式信息。如果没有,则进入其父类(如果存在的话),并对那些必须进行分析的方法开始进行处理。

4. 当 Introspector 类对 java.beans.BeanInfo 接口中的所有方法都从 BeanInfo 类中获得信息,或当已不存在可搜索的父类时,Introspector 类返回其结果。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

为使程序员的工作更轻松容易，Sun 公司提供了一个预先构造的类 java.beans.SimpleBeanInfo，对于所有 BeanInfo 方法它都返回一个 null 值。这样，就可以从这个类中继承并重载所选择的方法。程序清单 41.7 给出了 TextDisplay Bean 的 BeanInfo 类。注意我们是如何仅对 getPropertyDescriptors()方法进行重载的。父类对 java.beans.BeanInfo 接口中所有其它方法均返回 null 值。

```
import java.beans.*;

public class TextDisplayerBeanInfo extends SimpleBeanInfo {
    // override the getPropertyDescriptors method to provide that info.
    public PropertyDescriptor[] getPropertyDescriptors() {
      PropertyDescriptor[] properties = new PropertyDescriptor[4];

      try {
        properties[0] = new PropertyDescriptor(
"Text String", BeanClass, "getOutputText", "SsetOutputText" );
        properties[1] = new PropertyDescriptor(
                        "Text Color", BeanClass, "getFontColor","setFontColor");
        properties[2] = new PropertyDescriptor(
                        "Text Font", BeanClass, "getTextFont", "setTextFont" );
        properties[3] = new PropertyDescriptor(
                        "Background Color", BeanClass, "getBGColor", "setBGColor" );
      } catch( IntrospectionException e ) {
          return( null ); // exit gracefully if we get an exception.
      }
      return( properties );
    }
    private Class BeanClass = TextDisplayer.class;
}
```

让我们稍花费点时间查看一下程序清单 41.7 中的 try | catch 语句。注意在捕获 java.beans.IntrospectionException 异常时是如何返回一个 null 值的。如果捕获到这样的一个异常，通常意味着提供了一个不正确的 getter 或 setter 名，如果捕获到这种异常，应该总返回一个 null 值，以便 Introspector 类可以继续分析 Bean。读者应该能够扩展这个示例，对 java.beans.BeanInfo 接口中的其它方法进行重载。图 41.3 给出了 Sun 公司的 BeanBox 对 TextDisplay Bean 的 PropertySheet 窗口。注意属性的用户友好名是如何使用的，其父类的属性消失了，这也就达到了我们的目的。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

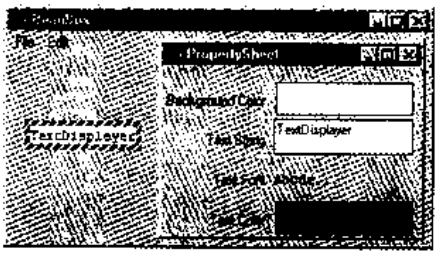


图 41.3　在 TextDisplay Bean 中属性使用用户友好名的 Sun 公司的 BeanBox 的 PropertySheet 窗口

41.7　定制:提供定制属性编辑器和 GUI 接口

到现在为止我们已经看到了如何创建 Bean,如何公开其属性、方法及事件、以及如何调整内省过程。读者可能已经从图中注意到,在 Bean 的属性中有一个 PropertyEditors(属性编辑器)。例如查看图 41.3,在 PropertySheet 窗口中,紧跟在"Text String"标签之后,有一个 TextField AWT 组件,其中已经填入了 OutputText 属性的值。我们不必为该组件提供任何代码,但是 Sun 公司的 BeanBox 是如何知道要提供它的呢? 答案是 BeanBox 应用程序寻问 java.beans.PropertyEditorManager,得到一个 java.lang、String 类型的对象的缺省 PropertyEditor,并将其显示出来。

PropertyEditor 和 Customizer 需要一个 GUI 环境,这并不意味着没有 GUI 环境 Bean 就不能运行。例如,一个为运行于服务器而设计的 Bean 有可能根本不使用(或不需要)GUI 环境。java.beans.Beans 类和 java.beans.Visibility 接口允许 Bean 在 GUI 和非 GUI 环境中有不同的行为。

41.7.1　属性编辑器和属性编辑器管理器

java.beans.PropertyEditorManager 类为大多数类型的 Java 类提供了缺省的属性编辑器(PropertyEditor)。如果仅仅使用本地(native)类型的 Java 数据和对象,则不存在任何问题,不需我们再做其它工作。但是如果所设计的 Bean 有一个属性不存在缺省的属性编辑器又怎么办呢? 每当设计自定义属性类型时,都会面临这一问题。对于不存在缺省属性编辑器的这种情况,我们必须自己提供。实际上,如果愿意,我们可以重新设计所有缺省属性编辑器,但是的确只在很少的情况下需要这样做,所以在这里我们不对这一问题进行讨论。这就是说我们必须通过在其类名中添加 Editor 提供一个附加类,供 PropertyEditorManager 使用。在大多数情况下,我们提供一个 java.awt.Component 的子类。然后,组件的属性单将弹出自定义的属性编辑器,供我们定制所需编辑的属性使用。此处,我们实际上并不需要设计自定义属性编辑器,因为大多数的 Bean 并不需要这样做,但是解释清楚如何设计还是必要的。设计一个属性编辑器所需的操作如下:

1. 自定义属性编辑器必须从 java.awt.Component 继承,以便能在属性单中显示出来。注意,这可能只是指继承一个 AWT 组件如:java.awt.TextField。

2. 自定义属性编辑器必须通过在属性类之后添加 Editor 方法派生其类名,除非它们在注册时与其容器的 PropertyEditorManager(属性编辑器管理器)注册在一起(见第 3 步)。例如,对于自定义属性类型 CustomProperty.class 的属性编辑器,必须命名为 CustomPropertyEditor.class。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

3. 对于不遵循第 2 步中标准命名规则的自定义 PropertyEditor，自定义属性类型必须通过调用 registerEditor()方法，将其自身与容器的 PropertyEditorManager 注册在一起。

4. 自定义属性编辑器必须一直击发 Propertychange 事件，以更新自定义属性。这一点是必须的！否则，容器将无法知道并更新组件。

读者可能会自问“我可以提供我自己的属性单吗?”回答是肯定的，并且对于复杂的 Bean，这也是绝对必要的。从本质上说，属性单是比较简单的，并且用户友好性也相对较差。下面一节我们将讨论如何重载属性单机制以提供自定义的对话框。

41.7.2 定制编辑器

所有应用程序构造器都必须实现一些置于其容器的自定义 Bean 的方法。这样就产生了属性编辑器(PropertyEditor)机制及属性单的思想。但是对于一个 Bean 有多种不同的方法定制，或有多个属性的这种特殊情况，又怎么办呢？对这一问题的解决办法是自定义(customizer)。Bean 开发者可以有选择地提供自定义类和用于标准属性单的 Bean。即使 TestReader Bean 的属性机制运转很好，我们在此也将创建一个自定义类，学习究竟如何创建这样的自定义类。

为实现一个自定义类，Bean 必须提供一个 BeanInfo 类。Bean 的自定义类的类名是通过调用 java. beans. BeanInfo 接口中的 getBeanDescriptor()方法而确定的。这与到目前为止我们所遇到的情况有些不同，对于自定义不存在缺省的内省设计模式，我们必须提供一个 BeanInfo，即使它只提供唯一的信息 BeanDescriptor。事实上，这正是我们对程序清单 41.8 中所给出的 TextReaderBeanInfo.class 所做的工作。注意这个类是对 java.beans.SimpleBeanInfo 的继承；父类实现了 java.beans.BeanInfo 类，并且我们重载了 getBeanDescriptor()方法，以便它能返回有含义的值。

```
import java.beans.*;

public class TextReaderBeanInfo extends SimpleBeanInfo {
    // override the getBeanDescriptor method to provide a customizer.
    public BeanDescriptor getBeanDescriptor() {
      return( new BeanDescriptor( BeanClass, CustomizerClass ) );
    }

    private Class BeanClass = TextReader.class;
    private Class CustomizerClass = TextReaderCustomizer.class;
}
```

虽然不存在设计模式，但习惯上命名一个自定义类时仍采用在类名后添加 customizer 的方法。注意我们将 TextReader 的自定义类命名为 TextReaderCustomizer.class。这是一个很

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

好的习惯。

在设计自定义类时，程序员有相当大的自由，只有两个限制：类必须继承 java.awt.Component，以便于它能放置于一个面板（Panel）或对话框（Dialog），类必须实现 java.beans.Customizer 接口。通过调用 SetObject()方法，给予自定义类一个对目标组件的引用。这以后，对大部分自定义类来说所要做的就是其自己的事了。然而要记住的是，我们必须（通过编译器）知道限制性属性，因为他们的访问方法可能产生 propertyVetoException 异常。最后，java.beans.customizer 接口包含 PropertyChangeListeners 的功能，由于 Bean 的容器可能将其自己作为一个侦听器与自定义类一起注册，任何属性更新都应随后有一个 firePropertyChangeListeners 调用，要做到这一点最简单的方法是使用一个 java.beans.PropertyChangeSupport 类，正如前面讨论绑定性属性时所做的那样。

程序清单 41.9 给出了 TextReaderCustomizer 类的大部分代码，为清晰起见，取掉了一些 AWT 特殊代码。完整的代码清单可以从 CD－ ROM 获得。现在来看一下 handleEvent()方法。该方法在用户输入数据时由 AWT 调用。注意如何强制捕获 setWidth()访问器所抛出的 PropertyVetoException 异常。我们还可以看到如何正确使用 PropertyChangeListener 方法。图 41.4 给出了在 Sun 公司的 BeanBox 内调用自定义类时的情况。



```
public class TextReaderCustomizer extends Panel implements Customizer {
    public TextReaderCustomizer() {
        setLayout( new BorderLayout() );
    }

    public void setObject( Object target ) {
      component = (TextReader)target;
      // generate the User Interface (code removed for clarity)
    }

    public boolean handleEvent( Event event ) {
      if ( event.id == Event.KEY-RELEASE && event.target == InputText ) {
        String old-text = component.getInputText();
        String text = InputText.getText();
        component.setInputText( text );
        changeAgent.firePropertyChange( "inputText", old-text, text );
      } else if ( event.id == Event.KEY-RELEASE && event.target == Width ) {
          int old-width, width;
          old-width = component.getWidth();
          try {
            width = Integer.parseInt( Width.getText() );
            try {
              component.setWidth( width );
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                changeAgent.firePropertyChange("width",
➡new Integer( old-width ), new Integer( width ) );
              } catch( PropertyVetoException e ) {
                // do nothing... wait for acceptable data.
              }
            } catch( NumberFormatException e ) {
              // do nothing... wait for better data.
            }
          }
          return ( super.handleEvent( event ) );
        }

        public void addPropertyChangeListener( PropertyChangeListener l ) {
          changeAgent.addPropertyChangeListener( l );
        }

        public void removePropertyChangeListener(PropertyChangeListener l) {
          changeAgent.removePropertyChangeListener( l );
        }

        private TextReader component;
        private TextField InputText, Width;
        private PropertyChangeSupport changeAgent = new PropertyChangeSupport( this );
    }
```

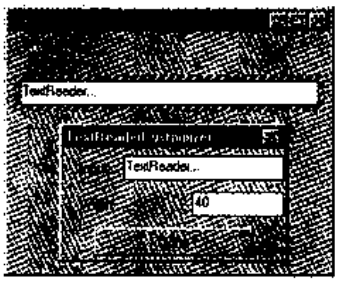


图 41.4 Sun 公司 BeanBox 显示 TextReader Bean 及其自定义对话框

41.7.3 在非 GUI 环境中提供另一种可选行为

不幸地是，对于一个 Bean 来说，GUI 接口并不总是可以使用的。最大可能的原因是 Bean 运行于后台或一个服务器。不管哪种情况，在非 GUI 环境下需要提供选择或附加行为的 Bean，可通过使用 java.beans.Beans 类和 java.beans.Visibility 接口来实现。

java.beans.Beans 类中的静态方法 isDesignTime()和 isGuiAvailable()，可用于检查 Bean 是否正在用于一个应用程序构造器，GUI 环境是否可以使用。如果 Bean 正使用于一个应用程序构造器，isDesignTime 方法返回 true（真）值，否则返回 false(假)值。如果 GUI 环境可使用，isGuiAvailable()方法返回 true(真)，否则返回 false(假)值。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

GUI 环境可以使用这一点并不是必须的，这意味着容器需要一个 Bean 来使用它。同样容器可能也需要知道一个 Bean 是否不在使用 GUI 环境，或者是否需要使用 GUI。通过实现 java.beans.Visibilty 接口，Bean 与其容器可以就这些信息进行通信。大多数的 Bean 不需要这种接口，也就不必去实现它，除非某个 Bean 打算使用它。该接口中有四个方法：

```
public abstract boolean avoidingGui()
```

该方法由一个容器调用，用来寻问一个 Bean 当前是否避开了 GUI 环境。对于当前正避开 GUI 环境的 Bean，该方法返回 true(真)值。注意这与明确指明它不需要 GUI 环境是不同的。例如，一个容器可能使用这种信息来释放正在被 GUI 环境所使用的资源，如果对该方法的调用返回 true(真)值的话。

```
public abstract void dontUseGui()
```

该方法由一个容器调用，用来通知 Bean，即使 GUI 环境可以使用，也不应该使用它。例如，在一个服务器上使用一个 Bean 容器，可调用该方法来告诉该 Bean 不可使用 GUI 环境。如果一个 Bean 选择按该方法处理(它应该如此)，那么该 Bean 对于随后的 avoidingGui()调用将返回 true(真)值。

```
public abstract boolean needGui()
```

该方法由一个容器调用，用来寻问一个 Bean 是否绝对需要一个 GUI 环境。如果一个 Bean 可在一个非 GUI 环境下运行，则应返回 false(假)值。注意，返回 true 然后不使用 GUI 环境是安全的，而返回 false 时又使用它则是不安全的。

```
public abstract void okToUseGui()
```

该方法由一个容器调用，用来告诉 Bean 有 GUI 环境可以使用并且该 Bean 能够使用它。该方法也可能在 dontUseGui()之后来调用，用来指示一个先前不可用的 GUI 环境又可用了。注意，调用该方法并不暗指一个 Bean 应该使用 GUI 环境，如果它没有打算使用的话。

41.8　企业级 JavaBeans

在使用 JavaBeans 框架的范围内一个新的发展是企业级(Enterprise)JavaBeans。企业级 JavaBeans 是在分布式多层次环境中建立和配置 Java 的组件模型。企业级 JavaBeans 扩展 JavaBeans 组件模型以支持服务器组件。

41.8.1　服务器组件

与标准的 JavaBeans 不同，企业级 JavaBeans 是为服务器组件而设计的。在服务器上运行组件的优点是它支持多层次结构。在一个多层次结构体系中，大部分的逻辑处理置于服务器，而不是客户机。

创建一个使用多层次设计的应用程序将使提高其可扩充性、其执行性能及其可靠性变得非常容易。使用企业级 Beans 组件可使用户开发出非常灵活的多层次应用程序。当业务规则和经济条件发展时，这些 Beans 将很容易修改。另外，与 RMI 组件一样，企业级 JavaBeans 可放置于任何地方，其处理过程独立于它所放置的位置。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

41.8.2 添加组件的"随处运行"功能

长期以来,Sun 公司一直在宣扬 Java 的"一旦编写,随处运行(write once,run anywhere)"的优点。使用企业级 JavaBeans,这一概念得到了扩展。现在不仅 Java 能够运行于任何平台,而且可以开发 Java 组件运行于任何执行组件的系统。环境将自动地把组件映射到服务器正使用的特定系统。所以,如果有一个服务器使用 DCOM,则企业级系统将映射到 DCOM。如果服务器正在使用 CORBA,它将映射到 CORBA,等等。

41.8.3 划分应用程序

现代面向对象的设计概括地将应用程序设计分为三个部分。这些部分为:用户界面、事务逻辑及数据。一般情况下,服务器是一个关系数据库管理系统(DBMS),应用程序与数据库管理系统进行通信,但数据库管理系统实际并非应用程序的一部分。然而在一个多层次的体系结构中,客户应用程序只包括用户界面编程。事务逻辑及数据被划分出去,移到一个或多个服务器上所配置的组件中。

将事务逻辑和数据移到服务器后,应用程序就可以充分利用多线程和多进程系统的优点。另外,服务器的组件在贯穿于用户应用程序的整个过程中,可以集中和共享并不充足的资源。随着系统需求的增加,频繁使用的企业级 Beans 可以在整个多重系统上复制和分发。这就是说对于一个多层次系统的可扩充性几乎没有限制,当系统资源枯竭时,总是可以复制另外一个系统以处理更多的负载。

另外,为证明提高了执行性能,这种复制可用于创建许多级别的冗余,这种冗余将帮助降低任何单点的失败。

41.8.4 可重用性及集成

对企业级 JavaBeans 的访问是通过一个定义完善的接口进行的。该接口允许使用企业级 Beans 创建可重用的软件构造块,就像一般的 JavaBeans 一样。一个函数一旦建立好,便可以在任何需要使用这一功能的应用程序中重复使用。

41.8.5 非可视组件

企业级 JavaBeans 是 JavaBeans 的一个服务器组件模型。它是运行于服务器的特殊化的非可视 JavaBeans。正如普通的 JavaBeans 一样,一个企业级 Bean 可以与其它的 Bean 汇编在一起,创建新的应用程序。

41.8.6 命名

企业级 JavaBeans 使用了 Java 的另一新特性——JNDI (Java Naming and Directory Interface)。JNDI 为映射专用系统名字到其实际计算机位置,定义了一种机制,很像 Internet 的域名系统,它允许我们将像 www. yahoo.com 这样的名字映射到该名字实际所代表的计算机系统。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

第 42 章　JavaIDL:Java 的 CORBA 接口

本章内容:

42.1　什么是 CORBA
42.2　Sun 公司的 IDL 到 Java 的映射
42.3　方法
42.4　创建基本的 CORBA 服务器
42.5　使用 JavaIDL 创建 CORBA 客户程序
42.6　在 CORBA 中创建回叫
42.7　CORBA 与现有对象的包装
42.8　在小程序中使用 CORBA

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

42.1 什么是 CORBA

通用对象请求代理结构(Common Object Request Broker Architeture,CORBA)是一个极具想象力的东西,它能使分布式对象在不考虑其位置和操作环境的情况下进行交互。CORBA仍还很不成熟,许多标准仍处于定义阶段,但是其庞大的基本结构已经定义。许多软件商仍在已经定义的某些特性上做工作。

CORBA 由若干个层组成。最低的一层为对象请求代理(Object Request Broker,ORB)。ORB 实际上是一种远程方法调用机制。ORB 是语言无关的,即我们可以使用任何语言创建对象,并使用 ORB 调用这些对象中的方法。我们也可以使用任何一种语言创建客户程序,通过 ORB 调用远程方法,这里存在一个对"任何一种语言"这一概念的理解问题。我们需要有一种定义在实现语言和 CORBA 的接口定义语言(IDL)之间的语言映射。

IDL 是一种描述语言——我们不可以使用它来编写工作程序,只能使用它来描述远程方法和远程属性。这种限制类似于 Java 中 Java 接口只能包含方法声明和常量的限制。

当从 IDL 转到其实现语言时,我们以实现语言产生了主干和框架。主干是客户与 ORB 之间的接口,而框架则是 ORB 与对象(或服务器)的接口。图 42.1 给出了 ORB、对象、以及希望调用该对象中方法的客户之间的关系。

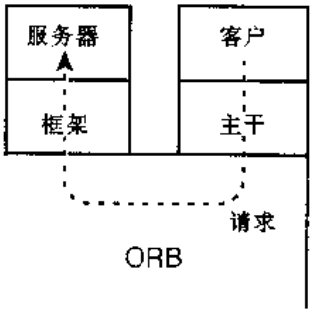


图 42.1 CORBA 客户使用 ORB 调用——一个 CORBA 服务器的方法

虽然 ORB 从概念上被划分为整个结构的一个独立部分,而实际上它常常只是应用程序的一部分。一个基本的 ORB 实现可能包括命名服务(稍后讨论),以及实现客户与服务器之间通信的一组库。一旦一个客户找到一个服务器,它将直接与那个服务器通信,并不再经过任何中间程序。这一点保证了 CORBA 实现的效率。

ORB 既是 CORBA 中最可视的部分,又是最乏味的部分。CORBA 的巨大效益来自于它所定义的所有服务。CORBA 中所定义的服务有:

- 生命周期(Lifecycle)
- 命名(Naming)
- 持续性(Persistence)
- 事件(Event)
- 事务(Transaction)
- 查询(Querying)
- 属性(Property)

这些服务是 CORBA 所定义的所有服务的一个子集。生命周期和命名服务明确了 Sun 公司的口号"网络就是计算机"。这些服务允许我们在不知道对象所驻留位置的情况下实例化新

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的对象。我们可能正在其自己的程序空间内创建一个对象，也可能正在世界的某处创建一个对象，我们的程序从来不知道这些。

生命周期服务允许我们在一个特殊系统上创建、删除、复制及移动对象。作为一个应用程序开发者，可能更想知道一个对象的驻留地，而作为一个系统程序员，更需要生命周期服务在实现这一定位时对应用程序程序员是透明的。在远程过程调用系统中，我们经常遇到的一个冲突是，在能够进行调用之前，所调用的服务器必须已经启动并处于运行状态。生命周期服务消除了这种冲突，它允许我们根据需要，在调用服务器中的方法之前，创建对象。

命名服务允许我们根据名字在网络上定位一个对象。我们需要能够在网络上移动而无需修改任何代码这样一种充分的灵活性。命名服务通过使一个对象与一个名字而不是网络地址发生联系，提供了上述功能。

持续性服务允许我们将对象保存在某处，并在以后检索这些对象。对象可能存放在一个文件中，也可能存放在一个对象数据库中，CORBA 标准不指定这一点，它将其留给了软件厂商。

事件服务是一个消息系统，它允许更为复杂的交互而不只是简单的消息调用。例如，我们可能使用事件服务实现一个基于网络的观察者可见的模型。其中有发送事件的事件提供者，有接收事件的事件消费者。服务器和客户既可以是主动的(push)，也可以是被动的(pull)。一个主动(push)服务器当其需要发送时发送出事件(它将事件推出(push)去)。而一个主动(push)客户有一个主动(push)方法，它使用该方法自动接收事件。一个被动(pull)服务器只有在得到请求时才发送出事件(我们必须将事件从服务器中拉(pull)出来)。一个被动(pull)客户只有在要求其接收时才接收事件。用“poll(投票)”代替此处的 pull(被动)，可能会更容易理解。一个被动(pull)服务器不会自己释放出事件，只有当向它投票时它才发出事件。一个被动(pull)客户会自己走出去为得到事件而投票。

事务服务是 CORBA 体系结构中最复杂的服务之一。它允许我们将跨越多个对象的操作定义成为单个事务。这种事务类似于数据库中的事务。它处理并发、锁定以及在操作失败时的回退。一个事务必须遵守一些核心要求，简写为 ACID：

原子性(Atomicity)：事务是一个单个事件。事务中的一切或者作为一个整体全部处理或全部不处理。不能按部分事务来处理。

一致性(Consistency)：在执行一个事务时，我们不能使系统处于一种不一致的状态。例如，如果我们还有一个航班的一个座位，当两个要求该座位的事务同时发生时，我们不能同时将这个座位指定给这两个不同的人。

隔离性(Isolation)：在事务被提交之前，其它人不应该看到事务的结果。即使是多个事务同时在执行，它们对于数据也有个先后顺序。

持续性(Durability)：如果提交了一个事务，则我们可以保证所做的改变已经完成，并存放在某处。所做的修改不会被丢失。

事务服务通常依赖于一个外部事务处理(TP)系统。

对象查询服务允许我们基于除名字以外的一些东西来定位对象。例如，我们可以定位所有在利比里亚注册的船只或所有位于乔治亚州的 Krispy Kreme 面包店。这种服务通常用于对象存放在对象数据库中的情况。

属性服务允许对象存放其它对象的信息。属性类似于一个粘贴便条。一个对象可能在一个粘贴便条上写下一些信息，并将其粘贴到其它对象上。这种服务有着巨大的潜力，因为它可

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使信息在某个对象不知觉的情况下与该对象联系起来。

完整 CORBA 系统的美妙之处在于，所有这些服务都可以通过 ORB 而获得，所以如果我们的程序可以与 ORB 交谈，则我们可以使用这些服务。当然，ORB 开发商可能尚未全部实现所有这些服务。

42.2 Sun 公司的 IDL 到 Java 的映射

为了在 CORBA 系统中使用 Java，我们需要一种标准的方法，将 IDL 中所定义的属性和方法转换到 Java 属性和方法。Sun 公司提出了一种映射并发布了一个从 IDL 定义产生 Java 主干和框架的程序。

在 IDL 中定义接口与在 Java 中定义接口类似，因为我们只定义方法的签名（参数及返回值），而并不定义方法的实现。

42.2.1 IDL 模块

IDL 模块与 Java 包等价。它将许多的接口集组织在一起放在其自己的名空间中。与 Java 包类似，IDL 模块也可以嵌套。下面是 IDL 模块定义的一个实例（其中未给出任何定义，稍后讨论）：

```
module MyModule{
    //在此处插入 IDL 定义，一个合法的 IDL 模块
    //必须至少有一个定义
};
```

这一模块将在 Java 中产生一个名为 MyModule 的 Java 包：

```
package MyModule;
```

对于嵌套的 IDL 模块，所生成的 Java 包也将是嵌套的。例如，考虑如下嵌套的模块定义：

```
module foo{
    module bar{
        module baz{
            //在此处插入定义
        };
    };
};
```

提示：不要忘记模块定义结束处大括号后面的分号（;）。与 Java、C 以及 C++ 不同，在 IDL 的大括号后必须使用一个分号。

baz 模块内 Java 包的接口定义可能是：

```
package foo.bar.baz;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

42.2.2　IDL 常量

与在 Java 中一样,在 IDL 中我们也可以定义常量值。IDL 常量定义的格式为:

```
const type variable = value;
```

常量的类型限制在如下类型之内:boolean、char、short、unsigned short、long、unsigned long、float、double 以及 string。

常量映射到 Java 时使用了一个特殊的方法。每个常量被定义成为一个类,该类只有一个静态、最终及公共的变量,变量名为 value,其值为指定常量。这是因为 IDL 允许在模块中定义常量,而 Java 则要求常量属于一个类。

下面是 IDL 常量定义的一个实例:

```
module ConstExample{
    const long myConstant = 123;
};
```

该 IDL 定义可能将产生如下的 Java 定义:

```
package ConstExample;
public final class myConstant{
    public static final int value = (int)(123L);
};
```

42.2.3　IDL 数据类型

IDL 数据类型基本上与 Java 的基本数据类型一样,但存在以下一些例外:

- Java 的 byte 类型对等于 IDL 的 octet。
- IDL 支持 String 类型,但称为 string。
- IDL 字符值只能在 0255 之间。JavaIDL 系统将检查字符值,保证字符值在这一范围之内,包括存放在字符串中的字符。
- IDL 支持 16 位、32 位以及 64 位的整数,但 32 位和 64 位类型的名字略有不同。在 IDL 中,32 位值称为 long 型,而在 Java 中 32 位值称为 int 型。Java 中的 long 型在 IDL 中对应为 long long 型。
- IDL 支持无符号 short、int 和 long 值。在 Java 中这些值存储在有符号变量中,我们在处理大的无符号值时必须小心,因为在 Java 中表示时最终有可能成为负数。

42.2.4　枚举类型

与 Java 不同,IDL 允许创建代表整型值的枚举类型。JavaIDL 系统把枚举类型转换为公共静态及最终值的类。

下面是 IDL 枚举类型的一个实例:

```
module EnumModule{
    enum Medals {gold, silver, bronze};
    };
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

这一定义将产生如下的 Java 类：

```
package EnumModule;
public class Medals {
    public static final int gold = 0;
                        silver = 1;
                        bronze = 2;
    public static final int narrow(int i) throws sunw.corba.EnumerationRangeException {
        if (gold <= i && i <= bronze) {
            return i;
        }
        throw new sunw.corba.EnumerationRangeException();
    }
}
```

由于允许我们声明枚举类型变量，JavaIDL 创建一个用来替代数据类型的持有者类。持有者类包含单一的实例变量 value，该变量持有枚举类值。Medals 枚举的持有者类为：

```
package EnumModule;
public class MedalsHolder
{
    //实例变量
    public int value;
    //构造器
    public MedalsHolder() {
        this(0);
    }
    public MedalsHolder(int _arg){
        value = EnumModule.Medals.narrow(_arg);
    }
}
```

我们可以通过传输枚举值到构造器的方式创建一个 MedalsHolder：

```
MedalsHolder medal = new MedalsHolder(Medals.silver);
```

narrow 方法用来进行值的范围检查，如果参数的值超出了枚举的范围，则产生一个异常。narrow 返回传送到它的值，所以可以使用该方法执行被动边界检查。例如：

```
int x = Medals.narrow(y);
```

如果在 Medals 枚举值范围内，则将 y 赋值给 x；否则，产生一个异常。

42.2.5 结构

IDL 的 struct 类似于一个不带方法的 Java 类。事实上，JavaIDL 将一个 IDL 的 struct 转换为一个 Java 类，该类仅有的一个方法为空构造器，一个为使用结构的所有属性的构造器。

下面是一个 IDL struct 定义实例：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
module StructModule{
    struct Person{
        string name;
        long age;
    };
};
```

这一定义将产生如下的 Java 类声明(一些 JavaIDL 指定的方法省略):

```
package StructModule;
public final class Person{
    //实例变量
    public String name;
    public int age;
    //构造器
    public Person() {}
    public Person(String _name, int _age){
        name = _name;
        age = _age;
    }
}
```

与枚举类型一样,struct 也产生一个代表该结构的持有者类。持有者类包含一个实例化变量 value。下面是 Person 结构的持有者类:

```
package StructModule;
public final class PersonHolder
{
    //实例变量
    public StructModule.Person value;
    //构造器
    public PersonHolder() {
        this(null);
    }
    public PersonHolder(StructModule.Person _arg){
        value = _arg;
    }
}
```

42.2.6　联合

联合(Union)是另一个没有移植到 Java 的数据结构。IDL 的联合实际上与 Pascal 的可变记录更相似,它需要一个"鉴别器"值。IDL 联合实际上是一组属性,某一时刻只有一个属性处于活动状态。鉴别器指示出当前正在使用的属性。一个简短的例子可以清楚地说明这一点,下面是一个 IDL 联合声明的例子:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
module UnionModule {
    union MyUnion switch (char) {
        case 'a': string aValue;
        case 'b': long bValue;
        case 'c': boolean cValue;
        default: string defValue;
    };
};
```

开关语句中的字符值就是鉴别器,它指示了联合中的三个变量中处于活动状态的那一个变量。如果鉴别器是'a',则 aValue 变量处于活动状态。由于 Java 没有联合,它被转换成为一个对每个不同变量有一个访问方法,对鉴别器有一个变量的类。这个类比较复杂,下面是 MyUnion 联合的 Java 类定义的一个子集:

```
package UnionModule;
public class MyUnion {
    //构造器
    public MyUnion() {
        //仅有一个空构造器
    }
    //鉴别器访问程序
    public char discriminator() throws
      sunw.corba.UnionDiscriminantException {
        //返回鉴别器的值
    }
    //分支构造器、获得和设置访问程序
    public static MyUnion createaValue(String value){
        //使用'a'鉴别器创建一个 MyUnion
    }
    public String getaValue() throws
      sunw.corba.UnionDiscriminantException {
        //返回 aValue 的值(只有当此时的鉴别器为'a'时)
    }
    public void setaValue(String value){
        //设置 aValue 的值,并设置鉴别器为'a'
    }
    public void setdefValue(char discriminator, String value)
      throws sunw.corba.UnionDiscriminantException {
        //设置 defValue 的值并设置鉴别器。虽然每个变量
        //都有一个这种形式的方法,但只有在联合中有一
        //个缺省值时该方法才有用。
    }
}
```

到现在我们应对持有者结构比较熟悉了。JavaIDL 对于联合也产生一个持有者结构。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

MyUnion 的持有者结构称为 MyUnionHolder,其中包含一个单一实例变量 value。

42.2.7　序列与数组

IDL 序列和数组都可以很清晰地映射到 Java 数组中。IDL 中的序列可以是无界(无上限)或有界(一个指定的上限)的。IDL 数组总是有固定大小的。由于 Java 数组有固定的大小,但上限在编译时未知,所以 JavaIDL 系统要在运行时进行数组检查,确保数组满足 IDL 模块中所定义的限制。

下面是一个 IDL 定义,其中包括一个数组、一个有界序列和一个无界序列:

```
module ArrayModule {
    struct SomeStructure {
        long longArray[15];
        sequence <boolean> unboundedBools;
        sequence <char, 15> boundedChars;
    };
};
```

在 Java 中对应的数组定义应为:

```
public int[] longArray;
public boolean[] unboundedBools;
public char[] boundedChars;
```

42.2.8　异常

CORBA 也有异常这一概念。与 Java 不同的是,异常不只是一种对象类型,它们是一些分散的实体。IDL 异常不能继承其它异常。除此之外,IDL 异常与 Java 异常一样,并且可以有实例变量。

下面是一个 IDL 异常定义实例:

```
module ExceptionModule {
    exception YikesError {
        string info;
    }
}
```

该定义将创建如下的 Java 文件(省略了一些 JavaIDL 特定的方法):

```
package ExceptionModule;
public class YikesError
    extends sunw.corba.UserException {
        //实例变量
        public String info;
        //构造器
        public YikesError() {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        super("IDL:ExceptionModule/YikesError:1.0");
    }
    public YikesError(String _info) {
        super("IDL:ExceptionModule/YikesError:1.0");
        info = _info;
    }
}
```

42.2.9 接口

接口是IDL重要的一部分。正如Java接口一样,一个IDL接口包括一套方法的定义。并且,一个IDL接口可以继承其它接口。下面是IDL接口定义的一个实例:

```
module InterfaceModule {
    interface MyInterface {
        void myMethod(in long param1);
    };
};
```

IDL方法的参数分为三类:in、out、inout。in参数与Java参数一样——是一个传递值的参数。即使是方法中可能改变变量的值,在该方法返回时,所做的改变也被丢弃。

out变量是一个只用来输出的变量。方法只设置这种变量的值,当方法返回时这种值被保留,但是没有任何值传送到该变量(它是无初始化的)。

inout变量是in和out的组合——可以传送值到方法;如果方法改变了该值,在方法返回时这种改变被保留。

Java参数的只输入(in)性使IDL到Java映射面临一个问题。然而,Sun公司已经提出了一种合理的解决办法。对任何out或inout参数,传送那个变量的持有者类。然后CORBA方法可以使用将要被返回的那个值,设置value实例变量。

42.2.10 属性

IDL允许我们在接口中定义变量。这些变量被转换为属性(Attributes)的get和set方法。属性可以指定为只读(readonly)属性,这可以防止对属性产生set方法。例如,如果定义一个如下的IDL属性:

```
attribute long myAttribute;
```

则Java接口中将包含如下的方法:

```
int getmyAttribute() throws omg.corba.SystemException;
void setmyAttribute() throws omg.corba.SystemException;
```

42.3 方法

在IDL中定义方法与在Java中声明方法类似,仅有少量变化。其中一个最值得注意的差别是CORBA支持可变参数的概念。换句话说,我们可以传送一个整形变量x到一个CORBA

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

方法,并且该方法可以修改 x 的值,当该方法返回时,x 具有修后的值。在普通 Java 方法中,x 将保持其原来的值。

在 IDL 中,方法的参数必须标记出其 in、out 或 inout 类型。in 参数不能被当前方法所改变,其工作方式与 Java 方法相同。out 参数表示当前方法将设置值,所传入的值被忽略。也就是说,如果参数 x 是一个 out 参数,CORBA 方法不能读取 x 的值,只能修改它的值。inout 参数可以被 CORBA 方法读,也可以被修改。

下面是使用 in、out、inout 参数的一个简单方法声明:

```
long performCalculation(in float originalValue,
    inout float errorAmount,out float newValue);
```

由于 Java 不支持改变参数的功能,Java - IDL 映射对于 out 和 inout 参数使用了特殊的持有者类。IDL 编译器已经产生结构和联合的持有者类。对于基本类型如 long 或 float,JavaIDL 具有形如 TypenameHolder 这样的固有持有者类。例如,long 型的持有者类为 LongHolder。每个这样的持有者类都包含一个公共的实例变量 value,该变量包含了参数的值。

IDL 和 Java 方法声明之间另一个主要的区别是其异常的声明方式。IDL 使用的是 raise 关键字而不是 throws。另外,异常列表要用括号括起来。下面是一个简单的方法声明,它产生若干个异常:

```
void execute() raise (ExecutionError,ProgramFailure);
```

42.4　创建基本的 CORBA 服务器

ORB 与服务器实现之间的接口称为框架(skeleton)。一个 IDL 接口的框架从 ORB 接收信息,激活相应的服务器方法,并向 ORB 发送回结果。一般情况下我们无需编写框架本身,只要提供远程方法的实现即可。

程序清单 42.1 给出了一个简单的银行接口的 IDL 定义。从中可以看出如何在 JavaIDL 中为这一接口创建一个客户和一个服务器。

程序清单 42.1　Banking 的声明代码

```
module banking {

    enum AccountType {
        CHECKING,
        SAVINGS
    };

    struct AccountInfo {
        string id;
        string password;
        AccountType which;
    };
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    exception InvalidAccountException {
        AccountInfo account;
    };

    exception InsufficientFundsException {
    };

    interface Banking {

        long getBalance(in AccountInfo account)
            raises (InvalidAccountException);

        void withdraw(in AccountInfo account, in long amount)
            raises (InvalidAccountException,InsufficientFundsException);

        void deposit(in AccountInfo account, in long amount)
            raises (InvalidAccountException);

        void transfer(in AccountInfo fromAccount,
            in AccountInfo toAccount, in long amount)
            raises (InvalidAccountException,InsufficientFundsException);
    };
};
```

42.4.1 编译 IDL 定义

在为自己的 CORBA 程序创建任何 Java 代码之前,我们必须首先将 IDL 定义编译成一组 Java 类。idlgen 程序读入一个 IDL 文件,并为创建在 IDL 文件中所定义的客户和服务器的各种接口、异常、结构、联合以及数据结构,创建类或支持类。

要编译 Banking.idl,idlgen 命令格式为:

```
idlgen -fserver -fclient Banking.idl
```

idlgen 允许我们使用 C 的预处理器包含其它文件,允许执行条件编译以及定义宏。如果我们没有 C 预处理器,可使用 -fno-cpp 选项,如下所示:

```
idlgen -fserver -fclient -fno -cpp Banking.idl
```

fserver 和 fclient 标志指示 idlgen 为创建服务器和客户分别创建类。然而,我们并不总是需要同时创建服务器和客户。如果我们正在创建一个 Java 客户,该客户将使用 CORBA 调用现存的 C++ CORBA 服务器上的方法,那么只需产生客户部分即可。如果我们正在使用 Java 创建一个 CORBA 服务器,而客户将使用其它语言,则只需产生服务器部分即可。

42.4.2 使用 IDL 的结构所定义的类

当一个 IDL 结构(struct)转换为 Java 类时,其中并没有自定义的 hashCode (散列码)和 e-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

quals 方法。这意味着包含相同数据的这种类的两个实例将会不相等。如果我们要向这些结构中添加自定义方法,必须创建一个单独的类,并定义转换一个类为另一个类的方法。

解决这一问题的一种办法是,创建一个包含与 IDL 结构相同信息的类,该类也包含 hashCode 和 equals 方法以及与所定义的 IDL 结构可以相互转换的转换方式。

程序清单 42.2 给出了一个 Account 类,其中包含与 Banking.idl 中所定义的 AccountInfo 结构相同的信息。

```
package banking;

// This class contains the information that defines
// a banking account.

public class Account extends Object
{
// Flags to indicate whether the account is savings or checking
    public String id;        // Account id, or account number
    public String password;        // password for ATM transactions
    public int which;        // is this checking or savings

    public Account()
    {
    }

    public Account(String id, String password, int which)
    {
        this.id = id;
        this.password = password;
        this.which = which;
    }
// Allow this object to be created from an AccountInfo instance

    public Account(AccountInfo acct)
    {
        this.id = acct.id;
        this.password = acct.password;
        this.which = acct.which;
    }

// Convert this object to an AccountInfo instance

    public AccountInfo toAccountInfo()
    {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        return new AccountInfo(id, password, which);
    }

    public String toString()
    {
        return "Account | " + id + "," + password + "," + which + " |";
    }

// Tests equality between accounts.
    public boolean equals(Object ob)
    {
        if (! (ob instanceof Account)) return false;
        Account other = (Account) ob;

        return id.equals(other.id) && password.equals(other.password) &&
                (which == other.which);
    }

// Returns a hash code for this object

    public int hashCode()
    {
        return id.hashCode() + password.hashCode() + which;
    }
}
```

42.4.3 JavaIDL 框架

当我们创建 CORBA 服务器时，IDL 编译器生成一个服务器框架(Skeleton)。该框架接收流入的请求，并确定出所要调用的方法。我们只需编写框架实际所要调用的方法即可。

JavaIDL 创建一个 Operations 接口，其中包含在 IDL 接口中所定义方法的 Java 版本。它还创建一个 Servant 接口，该接口扩展了 Operations 接口，框架类调用 Servant 接口的方法。换句话说，当我们创建实现远程方法的对象时，必须为其 IDL 定义实现 Servant 接口。

> **注意**：在通过一个单独对象可以实现的接口中定义远程方法，这种技术称为 TIE 接口。在 C++ 甚至早期的 Java ORBS 中，IDL 编译器产生一个实现远程方法的框架类。为了修改方法的实现，用户将创建框架类的一个子类，这种子类技术常称为基本对象适配器(Basic Object Adapter，BOA)。在 Java 中，TIE 接口的优点是，单个对象可实现多个远程接口。用户不能使用 BOA 对象完成这一任务，因为 Java 不支持多重继承。

例如，对于 Banking 接口的实现可以声明为：

```
public class BankingImpl implements BankingServant
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 42.3 给出实现 BankingServant 接口的完整 BankingImpl 对象。注意,每个远程方法必须声明产生一个 sunw.corba.Exception 异常。

```
package banking;

import java.util.*;

// This class implements a remote banking object. It sets up
// a set of dummy accounts and allows you to manipulate them
// through the Banking interface.
//
// Accounts are identified by the combination of the account id,
// the password and the account type. This is a quick and dirty
// way to work, and not the way a bank would normally do it, since
// the password is not part of the unique identifier of the account.

public class BankingImpl implements BankingServant
{
    public Hashtable accountTable;

// The constructor creates a table of dummy accounts.

    public BankingImpl()
    {
        accountTable = new Hashtable();

        accountTable.put(
            new Account("AA1234", "1017", AccountType.CHECKING),
            new Integer(50000));      // $ 500.00 balance

        accountTable.put(
            new Account("AA1234","1017", AccountType.SAVINGS),
            new Integer(148756));     // $ 1487.56 balance

        accountTable.put(
            new Account("AB5678", "4456", AccountType.CHECKING),
            new Integer(7742));       // $ 77.32 balance
        accountTable.put(
            new Account("AB5678", "4456", AccountType.SAVINGS),
            new Integer(32201)); // $ 322.01 balance
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// getBalance returns the amount of money in the account (in cents).
// If the account is invalid, it throws an InvalidAccountException

    public int getBalance(AccountInfo accountInfo)
    throws sunw.corba.SystemException, InvalidAccountException
    {

// Fetch the account from the table
        Integer balance = (Integer) accountTable.get(
            new Account(accountInfo));

// If the account wasn't there, throw an exception
        if (balance == null) {
            throw new InvalidAccountException(accountInfo);
        }

// Return the account's balance
        return balance.intValue();
    }

// withdraw subtracts an amount from the account't balance. If
// the account is invalid, it throws InvalidAccountException.
// If the withdrawal amount exceeds the account balance, it
// throws InsufficientFundsException.

    public synchronized void withdraw(AccountInfo accountInfo, int amount)
    throws sunw.corba.SystemException, InvalidAccountException,
      InsufficientFundsException
    {

        Account account = new Account(accountInfo);

// Fetch the account
        Integer balance = (Integer) accountTable.get(account);

// If the account wasn't there, throw an exception
        if (balance == null) {
            throw new InvalidAccountException(accountInfo);
        }

// If we are trying to withdraw more than is in the account,
// throw an exception

        if (balance.intValue() < amount) {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            throw new InsufficientFundsException();
        }

// Put the new balance in the account

        accountTable.put(account, new Integer(balance.intValue() - amount));
    }

// Deposit adds an amount to an account. If the account is invalid
// it throws an InvalidAccountException

    public synchronized void deposit(AccountInfo accountInfo, int amount)
    throws sunw.corba.SystemException, InvalidAccountException
    {

        Account account = new Account(accountInfo);

// Fetch the account
        Integer balance = (Integer) accountTable.get(account);

// If the account wasn't there, throw an exception
        if (balance == null) {
            throw new InvalidAccountException(accountInfo);
        }

// Update the account with the new balance
        accountTable.put(account, new Integer(balance.intValue() + amount));
    }

// Transfer subtracts an amount from fromAccount and adds it to toAccount.
// If either account is invalid it throws InvalidAccountException.
// If there isn't enough money in fromAccount it throws
// InsufficientFundsException.

    public synchronized void transfer(AccountInfo fromAccountInfo,
        AccountInfo toAccountInfo, int amount)
    throws sunw.corba.SystemException, InvalidAccountException,
        InsufficientFundsException
    {
        Account fromAccount = new Account(fromAccountInfo);
        Account toAccount = new Account(toAccountInfo);

// Fetch the from account
        Integer fromBalance = (Integer) accountTable.get(fromAccount);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// If the from account doesn't exist, throw an exception
            if (fromBalance == null) {
                throw new InvalidAccountException(fromAccountInfo);
            }

// Fetch the to account
            Integer toBalance = (Integer) accountTable.get(toAccount);
// If the to account doesn't exist, throw an exception
            if (toBalance == null) {
                throw new InvalidAccountException(toAccountInfo);
            }

// Make sure the from account contains enough money, otherwise throw
// an InsufficientFundsException.

            if (fromBalance.intValue() < amount) {
                throw new InsufficientFundsException();
            }


// Subtract the amount from the fromAccount
            accountTable.put(fromAccount,new Integer(fromBalance.intValue() - amount));

// Add the amount to the toAccount
            accountTable.put(toAccount,
                new Integer(toBalance.intValue() + amount));
        }
    }
```

42.4.4 服务器初始化

JavaIDL 是 Sun 公司对 IDL 到 Java 映射的实现，它与一个称为 Door ORB 的轻便的 ORB 一起发行。这种 ORB 只提供了可使客户与服务器相互交谈所需的功能，再没有其它别的功能。

根据所用 ORB 的不同，初始化及对象的激活方式也不相同。对于与 JavaIDL 一起发行的 Door ORB，初始化 ORB 使用下述代码行：

```
sunw.door.Orb.initialize(serverPort);
```

传送到 ORB 的 serverPort 参数是侦听流入的客户时将要使用的端口号，必须是一个整型数值。当客户与服务器连接时必须使用该端口号。

当初始化 ORB 之后，可以实例化实现对象。例如：

```
BankingImpl impl = new BankingImpl();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接着，创建框架并将实现对象传送给框架：

```
BankingRef server = BankingSkeleton.createRef(impl);
```

最后，通过发布对象名激活服务器：

```
sunw.door.Orb.publish("Bank",server);
```

程序清单 42.4 给出了银行服务器完整的 JavaIDL 启动程序。

程序清单 42.4　BankingServer.java 的源代码

```
package banking;

public class BankingServer
{

// Define the port that clients will use to connect to this server
        public static final int servicePort = 5150;

        public static void main(String[] args)
        {

// Initialize the orb
            sunw.door.Orb.initialize(servicePort);

            try {

                BankingImpl impl = new BankingImpl();
// Create the server
                BankingRef server = BankingSkeleton.createRef(impl);

// Register the object with the naming service as "Bank"
                sunw.door.Orb.publish("Bank", server);

            } catch (Exception e) {
                System.out.println("Got exception: " + e);
                e.printStackTrace();
            }
        }
}
```

42.5　使用 JavaIDL 创建 CORBA 客户程序

由于 IDL 编译器在服务器端创建了一个框架(Skeleton)类，用来接收远程调用请求，我们

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

可能希望在客户端也创建某种类型的框架,用来发送这些请求。的确可以,但客户端的类称为主干(stub)。

主干类实现了 Operation 接口(Skeleton 类所实现的同一 Operation 接口)。无论何时,当我们激活主干类的一个方法时,主干将创建一个请求并将其发送到服务器。

在主干的上面还有一个额外层称为引用。这种引用对象是我们用来进行服务器调用的对象,其名字为 IDL 接口名字后加上 Ref。

使用 JavaIDL 创建 CORBA 客户有以下两个简单步骤:

1. 使用特殊主干中的 createRef 方法创建一个主干的引用。

2. 使用 sunw.corba.Orb.resolve 方法,创建主干与 CORBA 服务器之间的连接。

为银行接口创建主干引用可使用下述代码行:

```
BankingRef bank = BankingStub.createRef();
```

接着,我们必须通过"解析"建立主干与 CORBA 服务器之间的连接。由于 JavaIDL 是所有 ORB 的标准 Java 接口,需要一个与 ORB 无关的命名模式。Sun 公司决定采用 URL 类型的命名模式,格式为:

```
idl:orb_name://orb_parameters
```

早期 JavaIDL 版本带有一个称为 Door ORB 的 ORB,它是一个轻便的 ORB,只包含一个命名模式。为了使用 Door ORB 访问一个 CORBA 对象,我们必须指定将要连接的 CORBA 服务器的主机名和端口号,以及所要访问对象的名字。这种信息的格式为:

```
hostname:port/object_name
```

如果我们想要使用 Door ORB 访问一个名为 Bank 的对象,该对象运行于一个本地主机服务器,所使用的端口号为 5150,则"解析"主干的方式为:

```
sunw.corba.Orb.resolve(
  "idl:sunw.door://localhost:5150/Bank",
  bank);
```

切记,bank 参数是 BankingStub.createRef 方法返回的 BankingRef。一旦主干得以"解析",我们便可以使用主干访问服务器中的远程方法。程序清单 42.5 给出了银行接口完整的 JavaIDL 客户程序。可以看出,一旦实现了主干与服务器的连接,就可以像调用本地对象一样调用主干中的方法。

```
import banking.*;

// This program tries out some of the methods in the BankingImpl
// remote object.

public class BankingClient
{
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
    public static void main(String args[])
    {

// Create an Account object for the account we are going to access.

        Account myAccount = new Account("AA1234", "1017", AccountType.CHECKING);

        AccountInfo myAccountInfo = myAccount.toAccountInfo();
        try {

// Get a stub for the BankingImpl object

        BankingRef bank = BankingStub.createRef();
            sunw.corba.Orb.resolve("idl:sunw.door://localhost:5150/Bank",bank);

// Check the initial balance
            System.out.println("My balance is:" + bank.getBalance(myAccountInfo));

// Deposit some money
            bank.deposit(myAccountInfo, 50000);

// Check the balance again
            System.out.println("Deposited $ 500.00, balance is: " +
                bank.getBalance(myAccountInfo));

// Withdraw some money
            bank.withdraw(myAccountInfo, 25000);

// Check the balance again
            System.out.println("Withdrew $ 250.00, balance is:" +
                bank.getBalance(myAccountInfo));

            System.out.flush();
            System.exit(0);

          } catch (Exception e) {
            System.out.println("Got exception:" + e);
            e.printStackTrace();
          }
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

42.6 在 CORBA 中创建回叫

在分布式计算中回叫(Callback)是一种方便的机制。当客户对某些事件需要得到通知而不是通过询问服务器该事件是否发生时,可使用回叫功能。在一般的 Java 程序中,我们只创建一个回叫接口,并传送给服务器一个实现该回叫接口的对象即可。当事件发生时,服务器将激活回叫对象中的方法。

可以看出,回叫在 CORBA 中就是这么简单。我们在 IDL 文件中定义一个回叫接口,然后在服务器接口中创建一个方法,以回叫接口作为其参数。下述 IDL 文件中定义了一个服务器接口和一个回叫接口:

```
module callbackDemo
{
    interface callbackInterface {
        void doNotify (in string whatHappened);
    };
    interface serverInterface {
        void setCallback (in callbackInterface callMe);
    };
};
```

在 JavaIDL 下,setCallback 方法将定义为:

```
void setCallback(callbackDemo.callbackInterfaceRef callMe)
    throws sunw.corba.SystemException;
```

一旦有了 callbackDemo. callbackInterfaceRef 对象,就可以随时激活 whatHappened 方法。在这一点上,客户与服务器处于对等地位,他们互为客户机和服务器。

42.7 CORBA 与现有对象的包装

在创建 CORBA 实现对象时,我们将对象与一个 CORBA 实现捆在了一起。当由 JavaIDL 系统所生成的 Servant 接口主动将我们的实现从 ORB 的细节分离时,我们的实现方法可产生 CORBA 的 SystemException 异常,将实现与 CORBA 连接在一起。这并不是一种理想的情况。

我们可以解决这一问题,但要预先做一些额外工作。首先,不使用 CORBA、RMI 或其它任何远程接口机制,集中精力实现所需要的对象。这将是我们所使用的所有的实现的一个拷贝。这个对象或一组对象可以定义其自己的类型、异常及接口。

其次,为了使这个对象可以远程使用,定义一个 IDL 接口,该接口要尽可能地与所能得到的对象的接口相近。可能会有它们并不完全一致的情况,但是我们可以注意到这些。

一旦从 IDL 定义生成了 Java 类后,就可以创建一个实现,该实现只是简单地激活实际实现对象的方法。这实际上与 TIE 接口是一回事,只有一个主要的例外——该实现类并没有涉及到 CORBA。我们也可以使用这一技术提供访问远程对象的多种方法。图 42.2 给出了一个可能提供的访问实现对象的各种方法的图示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

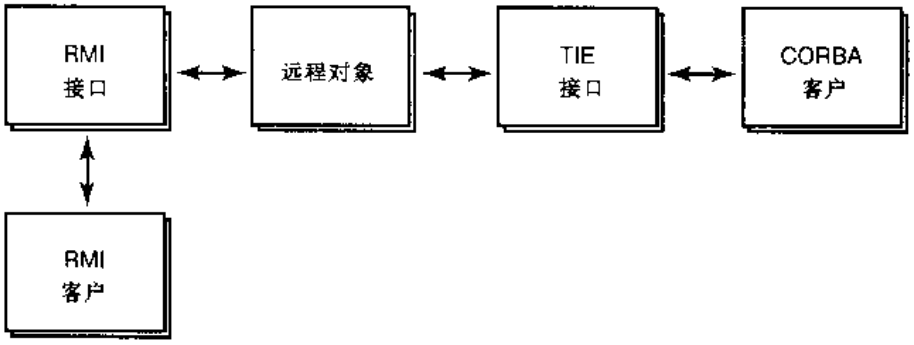


图 42.2　使用多种类型的远程对象系统访问单一对象

这看起来很简单,但还有一些复杂的东西必须处理。如果我们的实现对象定义了其自己的异常,我们必须将这些异常映射到 CORBA 的异常。还必须建立 Java 对象与 CORBA 定义的对象之间的映射。我们仍以银行系统接口为例,来说明从 CORBA 分离出应用程序时的问题和解决办法。

原来的银行接口定义了一个层次型的异常,即一个一般的 BankingException 异常,及其子类 InsufficientFundsException 和 InvalidAccountException 异常。这在 CORBA 中引出了一个问题,因为异常不能继承。我们必须在 IDL 文件中定义一个 BankingException 异常,如下所示:

```
exception BankingException{ };
```

另外,由于我们可能需要银行应用程序自身成为一个银行系统包,所以应把 IDL 模块名改为 remotebanking。

在 remotebanking 模块中 Banking 接口的实现必须执行两类映射。第一个,它必须将 Account 对象的实例转换成 AccountInfo 对象的实例。这看起来很让人头痛,实际上也确实如此,但这是必不可少的。如果我们一开始就将 CORBA 所定义的类与应用程序实现混在一起,那么应用程序最后还必须带上 CORBA 部分,即使它并不使用 CORBA。

42.7.1　与 CORBA 所定义类型的映射

我们可以定义静态的方法,来处理从应用程序数据类型到 CORBA 定义的数据类型之间的转换。例如,银行系统应用程序定义了一个 Account 对象,而 remotebanking 模块定义该对象为 AccountInfo。我们可以使用如下方法进行二者之间的转换:

```
//从一个 AccountInfo 对象创建一个 banking.Account
public static banking.Account makeAccount(AccountInfo info)
{
    return new banking.Account(info.id, info.password,info.which);
}
//从一个 banking.Account 对象创建一个 AccountInfo 对象
public static AccountInfo makeAccountInfo(banking.Account account)
{
    return new AccountInfo(account.id, account.password,
        account.which);
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

银行接口的远程实现需要访问真实的实现，所以 RemoteBankingImpl 对象的构造器需要一个 banking.BankingImpl 对象的引用：

```
protected banking.BankingImpl impl;
public RemoteBankingImpl(banking.BankingImpl impl)
{
    this.impl = impl;
}
```

42.7.2 创建远程方法包装

现在，所有我们的远程方法必须做的就是将任何流入的 AccountInfo 对象，转换为 banking.Account 对象，捕获异常并引发适当的远程异常。下面是远程方法 withdraw 的实现：

```
//在实际实现中调用 withdraw 函数，捕
//获异常并抛出对应的 CORBA 异常

public synchronized void withdraw(AccountInfo accountInfo,int amount)
  throws sunw.corba.SystemException, InvalidAccountException,
         InsufficientFundsException, BankingException
{
    try {

//调用实际的 withdraw 方法，首先将 accountInfo
//对象转换为 banking.Account 对象

        impl.withdraw(makeAccount(accountInfo), amount);

    } catch (banking.InvalidAccountException excep) {

//banking.InvalidAccountException 包含一个 Account 对象。
//当抛出 CORBA 异常时将其转换为 AccountInfo 对象

        throw new InvalidAccountException(makeAccountInfo(excep.account));

    } catch (banking.InsufficientFundsException nsf) {
        throw new InsufficientFundsException();
    } catch (banking.BankingException e) {
        throw new BankingException();
    }
}
```

如果能有 IDL 到 Java 的转换器自动产生这些内容，这当然更好。但如果这样，则实际的实现究竟是怎么回事就无法知道了。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

42.8　在小程序中使用 CORBA

虽然完整的 CORBA 套件非常庞大,但一个 CORBA 客户程序所需要的部分要小得多。对于客户端而言我们实际所需的全部东西就是 ORB。我们可以从网上的其它地方访问 CORBA 服务。这一点使得我们有了一个非常轻便的 CORBA 客户。换句话说,我们可以创建 CORBA 客户小程序。

在小程序中使用 CORBA 的唯一限制是,一个小程序只能与下载它的那个服务器建立网络连接。这就是说,所有的 CORBA 服务在 Web 服务器上必须可用(或必须有某种类型的代理设置)。

由于小程序不能对流入的网络连接进行侦听,所以小程序在大多数情况下不能是 CORBA 服务器。我们可能会找到一个 ORB,它通过使用小程序所建立的连接巧妙地避开了这些限制。目前,所使用的 Java ORB 大多数对回叫对象具有在小程序上运行 CORBA 的能力。对回叫,小程序可以创建一个本地服务器对象,然后将对其服务器对象的引用传送到运行于另一台机器上的一个 CORBA 服务器。然后 CORBA 服务器可作为客户通过引用激活小程序中的方法。

图 42.3 说明了如何使一个小程序作为一个 CORBA 服务器。

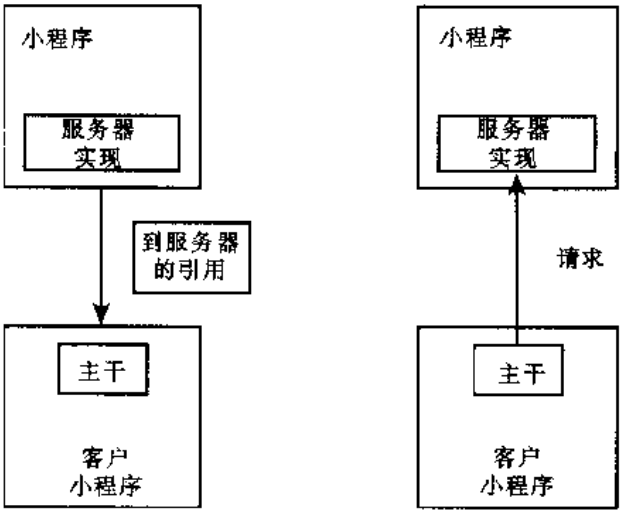


图 42.3　通过传送一个对本地 CORBA 服务器的引用,使小程序作为一个服务器使用

42.8.1　选择 CORBA 还是 RMI

CORBA 和 RMI 各有其优点和缺点。RMI 将成为客户端和服务器端 Java 的一个标准组件,成为一个功能优越而价格便宜的工具。由于它是纯 Java 的系统,可与 Java 的其它部分清晰地集成到一体。然而,RMI 仅仅是一个好的远程过程调用系统。

CORBA 定义了一个强劲的分布式环境,提供了分布式应用程序所需的几乎所有特性。大部分软件商并没有完全实现这些特性。大多数 CORBA 客户软件是免费提供,但服务器软件必须购买。这是当今大多数 Internet 软件所采用的价格模式。如果不需要 CORBA 的全部特性,并且也不想花费太多的钱,RMI 可能是最好的选择。

读者的所在公司可能认为 Java 尚未达到其"鼎盛时期"。如果是这种情况,而读者本人相

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

信 Java 定是未来的环境，那么可能的话应开始将 CORBA 纳入到目前的开发计划中。

CORBA 是一个独立于语言的系统。今天，我们可以使用许多 Java 的设计思想，在 C++ 下实现自己的应用程序。特别是，保持应用程序与用户界面的相互隔离，以及使软件尽量地模块化。如果我们在自己的系统组件之间使用 CORBA，则可以通过逐渐将各种组件替换成为基于 CORBA 的 Java 软件的方法，转移到 Java 上来。

如果读者是一个程序是员，并试图说明对 Java 的优越性持怀疑态度的管理人员，可使用 CORBA 在自己的应用程序中建立一个分布式接口（最好读者能有针对应用程序编写所使用语言的 CORBA 产品）。接着，使用 CORBA 编写一个 Java 小程序，实现其自己的应用程序的用户接口，以便与实际的应用程序进行交互。立刻，就可以将自己应用程序的一部分传送到每一个可以运行 Java 浏览器的平台上。最好该小程序能够像老的本地应用程序接口一样良好地运行。

这种相同的技术将使读者现有的 CORBA 应用程序开放到非传统的设备上，如蜂窝电话及 PDA。如果尚未准备好支持这些设备，至少现在已经有了一种途径。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 43 章　Java – COM 集成

本章内容：

43.1　一个重要的扩充
43.2　COM 概览
43.3　COM 接口定义
43.4　编译 ODL 文件
43.5　生成 GUID
43.6　使用 Java 创建 COM 对象
43.7　从 Visual Basic 调用 Java COM 对象
43.8　从 Excel 调用 Java 对象
43.9　从 Java 调用 COM 对象

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

43.1 一个重要的扩充

微软的 Java 虚拟机(JVM)可以在 Java 支持的 Internet Explorer 版本中找到,它包含了一些 Windows 专用的扩充,这些扩充不是标准 Java 的一部分。这些扩充中最重要的扩充之一是与组件对象模型(Component Object Model,COM)的集成。

警告:在微软的 Java 虚拟机中 Java - COM 集成是针对于特殊平台的。我们不能使用它开发平台无关的代码。任何使用 Java - COM 集成的代码肯定不能通过 100% 的纯 Java 测试。然而,如果我们所使用的是微软产品并且可以使用 Java,还是可以随意使用所有资源的。

43.2 COM 概览

许多人总觉得很难区分 OLE 和 ActiveX。事实上,有许多人认为 ActiveX 只是 OLE 的一个更迷人的名字而已。之所以造成这种混淆是因为 ActiveX 和 OLE 都是基于 COM 的。

COM 提供了对象之间的一种通信方式,而不管这些对象是在同一个程序、同一个机器上不同的程序,还是不同机器上的不同程序中。网络版的 COM 称为 DCOM (分布式 COM,Distributed COM),它是随着 NT4.0 的推出才形成其产品的。从微软的 Web 站点:http:\www.microsoft.com\oledev 可以得到一个对 Windows 95 的 DCOM 版本。DCOM 将成为未来 Windows 的一部分。

COM 目标不只是允许对象进行通信,它还鼓励开发人员创建可重复使用的软件组件。是的,这正是面向对象的开发所宣称的目标之一,但是 COM 远不只是一个典型的面向对象的编程语言如 Java。在 Java 或 C++ 中,我们重复使用一个组件是通过在程序中包含该组件而实现的。COM 允许我们重复使用组件时,组件仍处于其机器上,而不是成为用户程序的一个特殊部分。事实上,一个成功的 COM 对象可以由多个程序使用。关键是要有 COM 对象代码的一个拷贝,而不管有多少个不同的程序使用该对象。虽然我们可以对其 Java 类进行安排,使得只有某个特殊类的一个拷贝,但是这必须手工完成。使用 COM 时,这种重复使用将是自动实现的。

微软已经接受了基于组件的软件这一概念,并且已经为实现应用程序作为一个可重复使用的组件集,做出了巨大的努力。例如,Internet Explorer 就是以一个组件集的方式实现的,它只有一个很小的启动程序。例如,在 Internet Explorer 中 Java 虚拟机(JVM)组件就是一个单独的组件,可以被其它应用程序使用。

COM 最主要的基础是接口。如果理解了 Java 的接口,则对 COM 的接口也就理解了。在 Java 中,我们可以为一个类定义一组方法,然后可以通过宣称一个对象实现一个确定的接口来增加其它定义。换句话说,在 Java 中一个类所实现的方法的集合是由该类的定义和它所实现的接口共同决定的。COM 不支持类的概念,它只支持接口。由一个 COM 对象所实现的方法集合仅由其所实现的接口所决定。

每一个接口有一个唯一的标识称为它的 CUID (全局性唯一标识,Globally Unique Identifier),这是一个 40 位数字的数(160 位),这样生成它的原因是确保不会有两个相同的 CUID。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

COM 重要的一个方面是，当提供一个特殊接口（及其唯一 GUID）的软件发布后，则不应对该接口做任何修改。不能增加东西，不能删除东西，也不能修改其方法签名。如果有其它人正在使用这些组件，而该接口的下一版本进行了修改，则那些人的软件将不能再运行。所以如果需要修改一个接口，只好另外创建一个新的接口。

> **注意**：GUID 来自于 DCE RPC 标准，DCE RPC 中 GUID 称为 UUID（通用唯一标识，Universally Unique Identifiers）。

COM 接口是由 COM 的接口定义语言（Interface Definition Language，IDL）定义的，它是 DCE RPC IDL（一个现存远程过程调用标准）的一个超集。不要将 COM IDL 与 CORBA IDL 相混淆。虽然它们执行相同的功能，其语法却大不相同。也可以使用 OLE 的对象定义语言（Object Definition Language，ODL）来定义 COM 接口。微软提供了两种编译接口的不同编译器。编译 IDL 文件使用 MIDL，MIDL 包含在 Win32 SDK 或 Platform SDK 中。编译 ODL 文件使用 MKTYPLIB，MKTYPLIB 包含在 Visual J++ 或 ActiveX SDK 中。

访问一个 COM 对象有三种基本方式：作为一个过程中服务器，作为一个本地服务器，或作为一个远程服务器。当使用一个过程中服务器对象时，它运行在用户程序相同地址的空间内，作为一个本地服务器对象运行时在同一计算机上一个单独的程序运行，而一个远程服务器对象则运行于不同的计算机上。

不仅 COM 接口有一个唯一标识，COM 对象也有一个唯一标识。COM 对象的唯一标识称为它的 Class ID（CLSID）。一个 CLSID 实际上是一个 GUID，但它的作用有特殊目的，所以单独给它起了一个名字。CLSID 的值存放在 Windows Registry（注册表）文件中，用于寻找实现一个对象的特殊 DLL 或 EXE 文件。图 43.1 给出了一个用于 PowerPoint 应用程序的 CLSID 的 Registry 项。各种子键字如 LocalServer 和 InprocHandler 指示出当 PowerPoint 对象作为一个本地服务器或一个过程中服务器使用时，所使用的 DLL 或 EXE 文件。

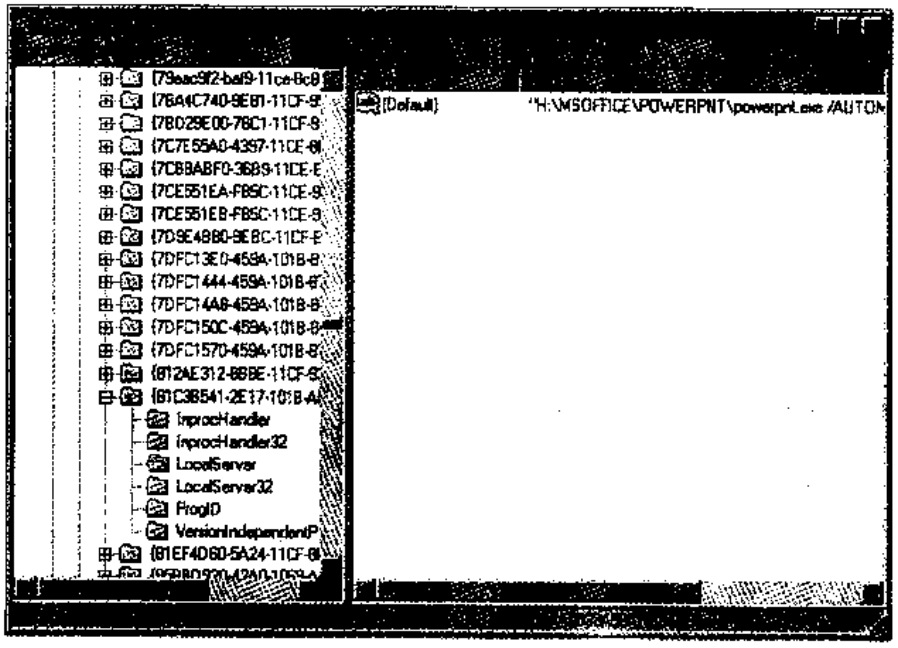


图 43.1　一个 CLSID 的 LocalServer（本地服务器）的 Registry 项，指示提供一特殊 COM 接口的 EXE 文件

除了 OLE 和 ActveX 实际所执行的功能外，它们之间存在的唯一区别是它们各自使用不

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

同的接口。所有 OLE 和 ActveX 接口均是使用 COM 定义并实现的。

43.3 定义 COM 接口

要建立一个 COM 对象,必须首先使用 IDL 或 ODL 创建一个接口。由于 MKTYPLIB 工具(ODL 编译器)来自于 Visual J++ 中,本章所讲的实例使用 ODL 而不是 IDL。

程序清单 43.1 给出了一个样本 ODL 文件。在这个文件中有许多东西看起来比较陌生。但是通过一一分析之后,就会知道这些东西并不像看起来的那样复杂。

```
// JavaObject.odl

// First define the uuid for this type library
[
  uuid(D65E5380-6D58-11d0-8F0B-444553540000)
]

// Declare the type library

library LJavaObject
{

// Include the standard set of OLE types
    importlib("stdole32.tlb");

// Define the uuid for an odl interface that is a dual interface
// A dual interface is the most flexible because it supports the
// normal interface calling mechanism and also dynamic calling.
//
    [ odl, dual, uuid(D65E5381-6D58-11d0-8F0B-444553540000) ]

// Declare the IJavaObject interface (dual interfaces must inherit
// from the IDispatch interface)

    interface IJavaObject : IDispatch
    {
// Declare the reverseString method that takes a string as input
// and returns a string
        HRESULT reverseString( [in] BSTR reverseMe,[out, retval] BSTR * reversed );

// Declare the square method that takes an integer and returns
// an integer
        HRESULT square( [in] int squareMe,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        [out, retval] int * squared );
    }

// Declare a class that implements the IJavaObject interface
    [ uuid(10C24E60 - 6D5D - 11d0 - 8F0B - 444553540000) ]
    coclass JavaObject
    {
        interface IJavaObject;
    }
};
```

首先，当使用 ODL 创建一组接口时，我们将其编译为一种类型库。一个类型库是对一个 ODL 文件而言的，就像一个 .CLASS 文件是对 Java 源文件一样。一个类型库必须有其自己的 GUID，下面的语句声明了一个库及其 GUID(切记 GUID 是 UUID 的另外一个名字)：

```
[
uuid(D65E5380 - 6D58 - 11d0 - 8F0B - 444553540000)
]

library LJavaObject
```

> **注意**：这看起来有些笨拙，但是定义一个对象的 GUID 只要在该对象的前面即可。在其它语言中，通常要与对象本身一起开始。

importlib 语句与 Java 中的 import 关键字类似。在上一例子中，导入了一组标准的 OLE 定义。在 importlib 之后是 IJavaObject 接口的定义：

```
[odl, dual, uuid(D65E5381 - 6D58 - 11d0 - 8F0B - 444553540000)]

interface IJavaObject : Idispatch
{
```

注意，uuid 关键字是与 odl 和 dual 关键字一起使用的。通常用来定义 uuid 的用括号括起来的区域可用于任何类型的属性。我们总可以在这里找到 uuid，因为每个接口和类必须有其自己的唯一标识。

当我们使用 ODL 定义一个接口时，要使用 odl 关键字。dual 关键字说明接口是一个双重接口。

COM 激活方法有两种不同的方式——通过一个查找表或通过一个发送接口。查找表是大家比较熟悉的 vtable，它是一个虚拟方法查找表，类似于 C++ 中的 vtable 表。发送接口允许我们执行动态方法调用。当我们使用一个发送接口时，在方法被激活之前，会发生一个附加层查找。这使得使用发送接口调用方法比使用 vtable 调用方法要慢一些，但这种方法对于解释性语言如 Visual Basic 很有用。为了获得最大的灵活性，在实现类时通过将其声明为双重性

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接口,使其同时可使用 vtable 和发送接口。

方法的定义看上去也很奇怪:

```
HRESULT reverseString([in]BSTR reverseMe,
    [out, retval]  BSTR   * reversed);
```

不管相信还是不相信,reverseString 方法确实返回一个字符串,而不是我们所看到的声明中的 HRESULT 值。当创建这一双重性接口时 HRESULT 返回值是必要的。实际的返回值是通过为参数之一指定[out,retval]属性指定的。具有[in]属性的参数是一个输入参数,而具有[out]属性的参数是输出参数。

BSTR 数据类型是一种基本字符串,这也是 COM 中表示字符串的一般方式。表示字符串还有其它的方法,但 BSTR 与 OLE 及 Visual Basic 兼容。我们可能会非常惊讶,Java - COM 编译器将 reverseString 的定义转换成了如下所示的非常简单的方法声明:

```
public String reverseString(String reverseMe)
throws com.ms.com.ComException
```

ODL 文件底部的 JavaObject 类的定义说明了一个 JavaObject 类所实现的接口。本例中只有一个接口:IJavaObject。如果查看一下系统中可使用的不同类,特别是 OLE 服务器,就会发现大部分的类实现了多个不同的接口。

43.4 编译 ODL 文件

Visual J++ 和 ActiveX SDK 中包含的 MKTYPLIB 程序,将一个 ODL 文件编译成一个类型库文件,其扩展名为 .TLB。编译 JavaObject.odl 的 MKTYPLIB 命名格式为:

```
mktyplib JavaObject.odl
```

缺省情况下,MKTYPLIB 使用预处理器允许我们使用 # include 和其它预处理器指令。不幸的是只有在我们有 C 预处理器的情况下这样做才有效。如果没有预处理器,则必须使用 /nocpp 选项,如下所示:

```
mktyplib /nocpp JavaObject.odl
```

43.5 生成 GUID

JavaObject ODL 文件包括三种不同的 GUID。我们不必自己提供这些值(事实上也不应该提供)。实际上 Visual C++ 和 ActiveX SDK(可能还有其它的软件包)带有一个称为 GUIDGEN 的工具,该工具随机地产生这些值,并以几种方式格式化这些值,甚至可以将这些值拷贝到剪切板(Clipboard)上,这样我们就可以将其粘贴到自己的源代码中。图 43.2 给出了一个 GUIDGEN 的对话样本。

43.6 使用 Java 创建 COM 对象

为了创建一个Java对象,实现一个或多个COM接口,我们需使用JAVATLB命令创建特

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

殊的包装类。例如，为了给 JavaObject.tlb（由 JavaObject.tlb 编译产生）中的信息创建包装

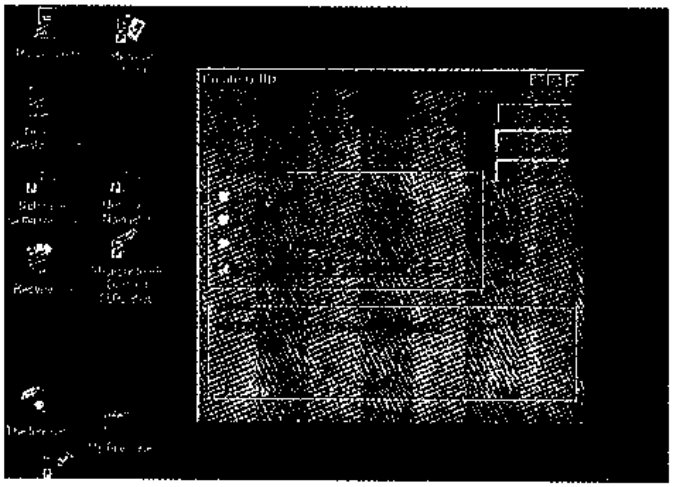


图 43.2　GUIDGEN 工具为用户自动产生 GUID 值

（Wrapper）类，可使用 JAVATLB 命令如下：

```
javatlb JavaObject.tlb
```

对 JavaObject.tlb 文件，JAVATLB 创建一个称为 IJavaObject 的接口和一个称为 JavaObject 的 Java 类。这些类属于 javaobject（全部小写）包，并且放置于 \ WINDOWS \ JAVA \ TRUSTLIB 目录下。记住，每个包都有其自己的子目录，所以如果查看 \ WINDOWS \ JAVA \ TRUSTLIB 目录，将会看到一个 javaobject 目录，其中存放着 IJavaObject.class 和 JavaObject.class。

当包装类创建之后，我们只要填写适当的方法即可。程序清单 43.2 给出了在 IJavaObject 接口中用来实现方法的 JavaObjectImpl 类。

程序清单 43.2　JavaObjectImpl.java 的源代码

```
import com.ms.com.*;
import javaobject.*;

public class JavaObjectImpl implements IJavaObject
{
    public String reverseString(String in)
    throws ComException
    {
        StringBuffer buff = new StringBuffer();

// Start at the end of the input string and add characters
// to the string buffer. This puts the reverse of the string
// into the buffer.

        for (int i=in.length()-1; i >= 0; i--) {
            buff.append(in.charAt(i));
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }

    // Return the contents of the buffer as a new string
        return buff.toString();
    }

    public int square(int val)
    throws ComException
    {
    // Return the square of val
        return val * val;
    }
}
```

当编译完 JavaObjectImpl 后(必须使用微软的编译器 JVC),使用 JAVAREG 工具将关于 JavaObjectImpl 的信息放置到系统注册表中。COM 将使用注册表定位 COM 对象,并找出为一个特定的对象运行服务器的方式。下面的命令注册 JavaObjectImpl,并给出 JavaObject 的 ProgID:

```
JAVAREG /register /class:JavaObjectImpl /ProgID:JavaObject
```

警告:在 JAVAREG 命令中一定不能在 JavaObjectImpl 后放置 .class。我们要给 JAVAREG 的是类的名字,而不是包含该类的文件名。

ProgID 值是一个简单的名字,其它程序如 Visual Basic 可以使用它来定位 JavaObject 类。JAVAREG 在注册表的 HKEY_CLASSES_ROOT 部分创建一个项为 JavaObject,它包含了一个称为 CLSID 的子键字,该子键字包含 JavaObjectImpl 的类 ID(GUID)。图 43.3 给出了由 REGEDIT 命令所显示的这一注册表项。

JAVAREG 在 HKEY_CLASSES_ROOT 中的 CLSID 下还创建了一个项,该项的键字为 JavaObjectImpl 的 CLSID(包含在 JavaObject 的 ProgID 项中的同一个 CLSID)。图 43.4 给出了由 REGEDIT 所显示的 CLSID 所建立的项。

使这个类能被世界上其它所有人使用的最后一步是拷贝 JavaObjectImpl.class 文件到 \WINDOWS\JAVA\TRUSTLIB 目录下。

注意:如果 Windows 95 或 Windows NT 安装到其它目录下而不是 \WINDOWS 目录,应使用相应的那个目录名再加上 \JAVA\TRUSTLIB。例如,如果是在 Windows NT 下运行,而 Windows NT 安装在 \WINNT 目录下,则把类文件拷贝到 \WINNT\JAVA\TRUSTLIB 目录下。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

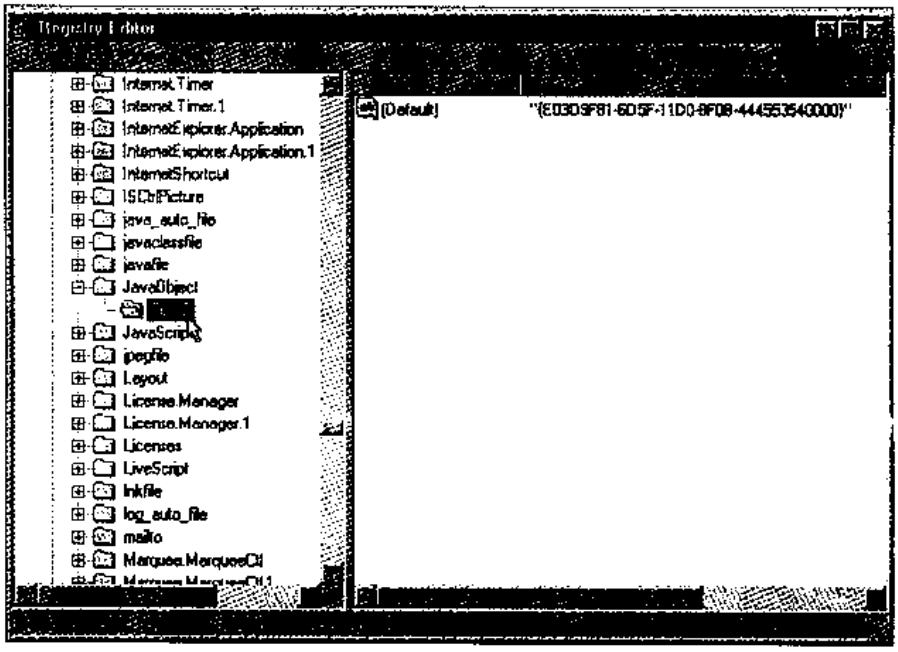


图 43.3　ProgID 映射一个简单的文本名到一个 160 位的 CLSID 值

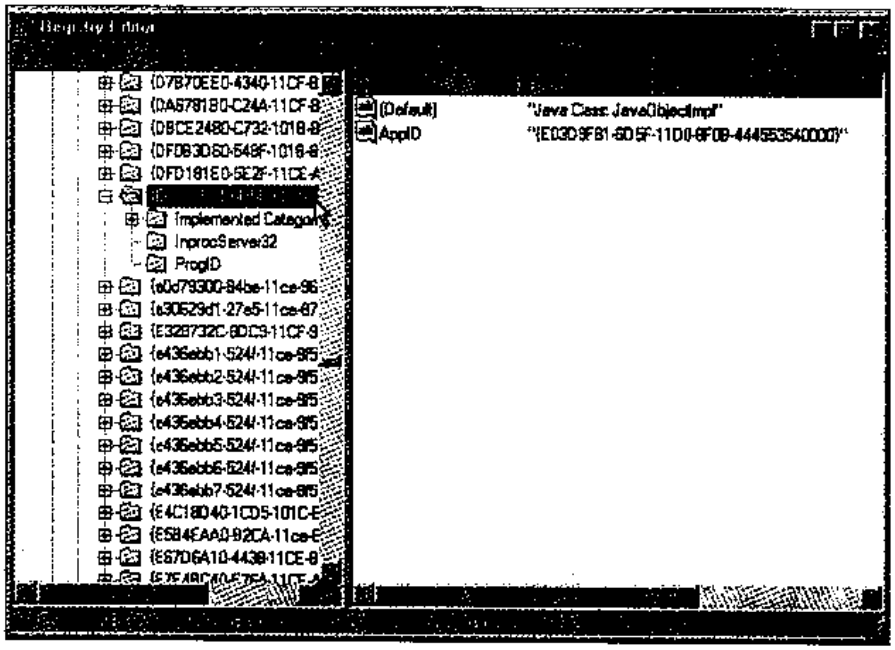


图 43.4　JAVAREG 在 CLSID 键字下生成的项

43.7　从 Visual Basic 调用 Java COM 对象

如果已经运行 JAVAREG 对所建立的类进行了注册，并且已经把该类拷贝到了 TRUSTLIB 目录下，现在应该能够从其它程序中访问这个类了。我们可以创建一个简单的 Visual Basic 应用程序访问这个类。在 VB 应用程序的声明部分加入下列语句：

```
Dim javaob as Object
```

然后是 VB 应用程序启动时所调用的 Form _ Load 子程序，如下所示：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Private Sub Form_Load()
Set javaob = CreateObject("JavaObject")
End Sub
```

JavaObject 字符串是对象的 ProgID。如果在运行 JAVAREG 时使用了其它字符串作为 ProgID,则此处应使用相应的字符串。

现在我们可以使用 JavaObject 类中的方法了。在该实例 VB 应用程序中,有两个文本字段: Text1 和 Text2。下面的子程序使用 Text1 中的文本,通过运行 JavaObject 中的 reverseString 方法,将其结果文本放在 Text2 中:

```
Private Sub Text1_Change()
Text2.Text = javaob.reverseString(Text1.Text)
End Sub
```

图 43.5 给出了动作中的这一 Visual Basic 应用程序。

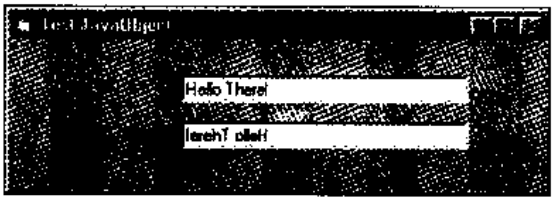


图 43.5 一个可以使用 Java 对象的 Visual Basic 应用程序

43.8 从 Excel 调用 Java 对象

Microsoft Excel 及其它 Microsoft Office 产品均有其自己的内置式 Visual Basic 版本。这就是说,我们也可以从 Excel 中访问 Java 对象。

警告:要访问 Java 对象必须使用 32 位的 Excel 版本。16 位的 Excel 版本不能使用 32 位的 COM 访问,也不能访问 Java 对象。下面的例子使用的是 Windows 95 Office 套件中的 Excel 7.0a。

为了创建一个 Excel 函数,启动 Excel 并从主菜单中选择 Insert 、Macro 、Module 命名,如图 43.6 所示。

该函数必须访问 JavaObject 类,然后调用 reverseString。程序清单 43.3 给出了函数 ReverserS 。

```
Function Reverser$ (reverseMe$ )
    Dim javaob As Object
    Set javaob = CreateObject("JavaObject")
    Reverser$ = javaob.reverseString(reverseMe$ )
End Function
```

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友, Java群空间; 可以联系群主获取更多大型企业内部技术教程。

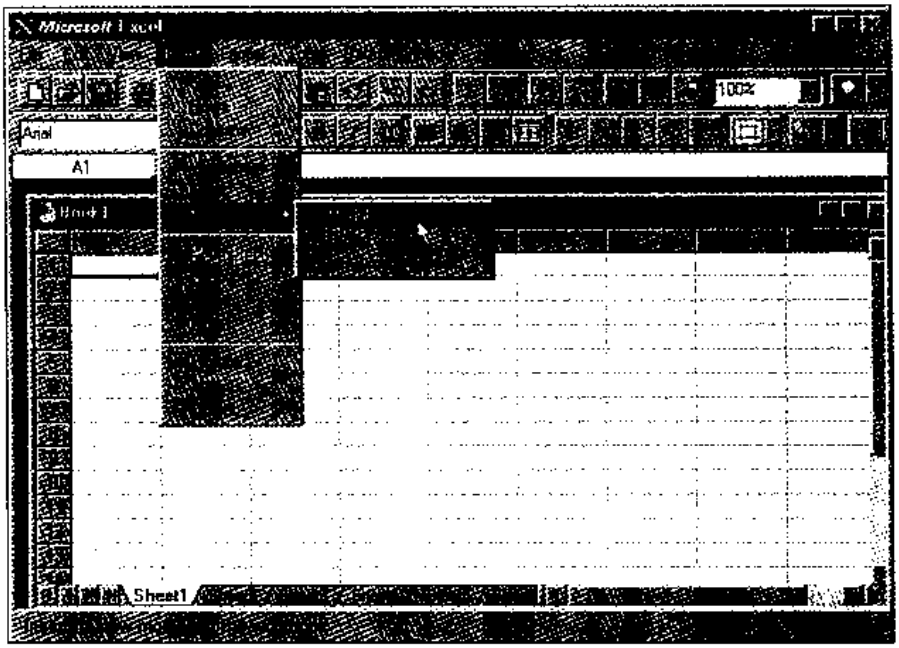


图 43.6　为了创建一个 Excel 函数，需要插入一个代码模块

> **注意**：要保证所使用的 Microsoft Java SDK 是最新版本。最早的版本使用 COM 集成时存在问题。

当定义了这一函数后，就可以在数据表中使用它了。例如，假设我们要使用数据表中 A1 单元中的信息，翻转该信息并将结果放在 A2 单元中。只需进入 A2 单元并输入以下公式：

```
= Reverser(A1)
```

注意，此处的 Reverser 后不使用 $。现在，在 A1 中所输入的任何文本将自动翻转并出现在 A2 中。图 43.7 给了一个样本数据表。

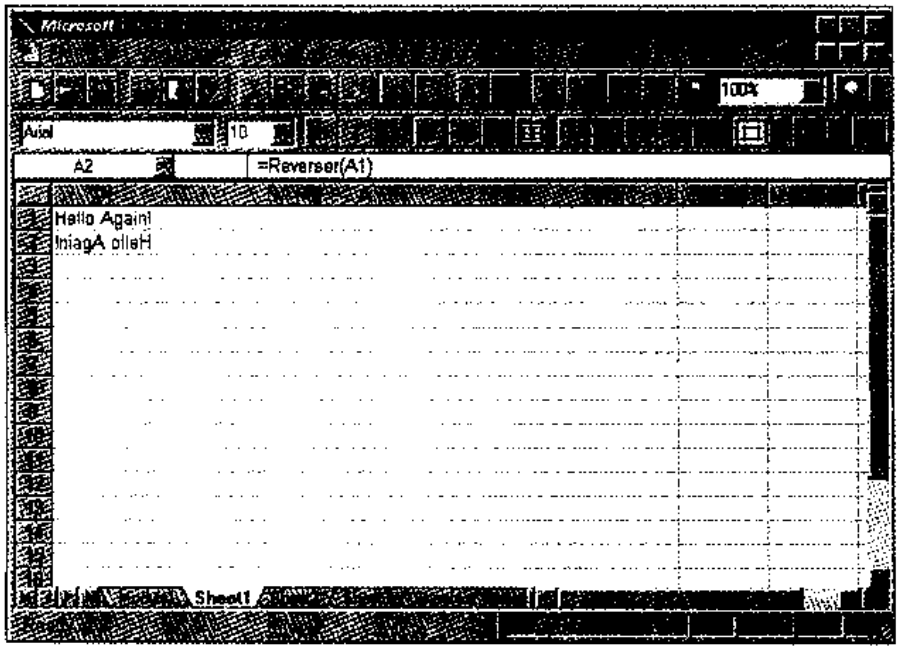


图 43.7　Excel 可以使用 Java 对象执行所感兴趣的功能

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

43.9 从 Java 调用 COM 对象

正如可以通过 COM 访问 Java 对象一样，Java 对象也可以访问其它 COM 件对象。这实际上为我们打开了 Windows 平台的各种可能性。如果我们想要创建一个图表，则可以使用 Excel 的图表功能。如果我们想要创建一个排版整洁的打印输出，可在 Word 中创建一个文档并打印输出。最重要的是我们不必进行创建 ODL 文件这一叫人头痛的工作，只要能得到所要使用的应用程序类型库即可。

在 Microsoft Word 7.0 版中，我们可以免费从微软的站点：http://www.microsoft.com/WordDev/Articles/wd70endl.htm 中获得类型库。

当有了类型库后，对该库运行 JavaTLB，创建 Java 包装。例如，对 Word 7.0 的类型库，JAVATLB 命令为：

```
javatlb wb70en32.tlb
```

该命令创建一个可使用的 WordBasic 接口类。由于 Word 以本地服务器对象运行，要访问它还必须做一些工作。

如果给定了对象的 CLSID，LicenseMgr 对象可以访问一个本地服务器对象。我们不必自己查看 Word.Basic CLSID 并将其放到源代码中。相反，我们可以从其程序中访问系统 Registry 并在运行时找到 CLSID。

Registry 类给我们提供了访问注册表的功能。给定一个 RegKey 对象，我们可通过使用父键字调用 RegKey 构造器检索子键字。例如，我们想要知道 CLASSES_ROOT \ Word.Basic \ CLSID 的键字。那么首先需要“CLASSES_ROOT”的 RegKey。由此创建一个 Word.Basic 的 RegKey，然后使用 Word.Basic 的 RegKey 创建 CLSID 的 RegKey。

当有了所需的 RegKey 后，通过调用 enumValue 方法我们可以访问缺省值。程序清单 43.4 给出了一个演示程序，该程序运行 Word 7.0，创建一个简单的“Hello World”文档并打印出来。

```
import wb70en32.*;
import com.ms.com.*;
import com.ms.lang.*;

public class WordDemo extends Object
{
    public static void main(String[] args)
    {
        try {

// Get the Registry Key for CLASSES_ROOT
            RegKey root = RegKey.getRootKey(RegKey.CLASSES_ROOT);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// From CLASSES_ROOT, get the key for Word.Basic
            RegKey wbkey = new RegKey(root,"Word.Basic", RegKey.KEYOPEN_READ);

// From Word.Basic, get the CLSID
            RegKey clsid = new RegKey(wbkey, "CLSID",RegKey.KEYOPEN_READ);
// Retrieve the CLSID from the CLSID key (it's the default value)
            String classID = ((RegKeyEnumValueString)clsid.
                enumValue(0)).value;

// Create a License Manager for accessing local server objects
            ILicenseMgr lm = (ILicenseMgr) new LicenseMgr();

// Get a reference to WordBasic
            WordBasic wb = (WordBasic)
                lm.createInstance(classID,
                    null, ComContext.LOCAL_SERVER);

// Create a new file
            wb.FileNewDefault();

// Insert some text
            wb.Insert("Hello World!");
            wb.InsertPara();
            wb.Insert("Hi there!");

// Print the text
            wb.FilePrintDefault();
        } catch (Error e) {
            e.printStackTrace();
        }
    }
}
```

提示：切记，在 MS Java 环境下使用 jview 命令运行程序，而不是使用 java。

如果我们想查看 WordBasic 对象中的方法，可使用 Visual J++ 中的 OLE 对象观察器(OLE2VIEW)，或 ActiveX SDK 中的 OLE 对象观察器(OLEVIEW)。图 43.8 给出了 OLE2VIEW 显示 WordBasic 对象中的方法之一的情况。

在使用 WordBasic 对象或其它对象时我们肯定会遇到的一个问题是，有些方法使用了变异类型的参数。Java－COM 包中有一个 Variant 对象，它允许我们传送变异类型的参数。例如，如果我们想要调用一个使用了两个变异类型参数的方法，可创建两个 Variant 对象的实例。变异参数在参数为可选时很有用。在这一例子中，假设第二个参数为可选参数，第一个参

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

数为整型参数。则事件的顺序将如下所示：

```
Variant p1 = new Variant();  //创建第 1 个参数
p1.putInt(5);                //使参数值等于 5
Variant p2 = new Variant();  //创建第 2 个参数
p2.noParam();                //该参数不传送值
SomeObject.funMethod(v1,v2);//调用方法
```

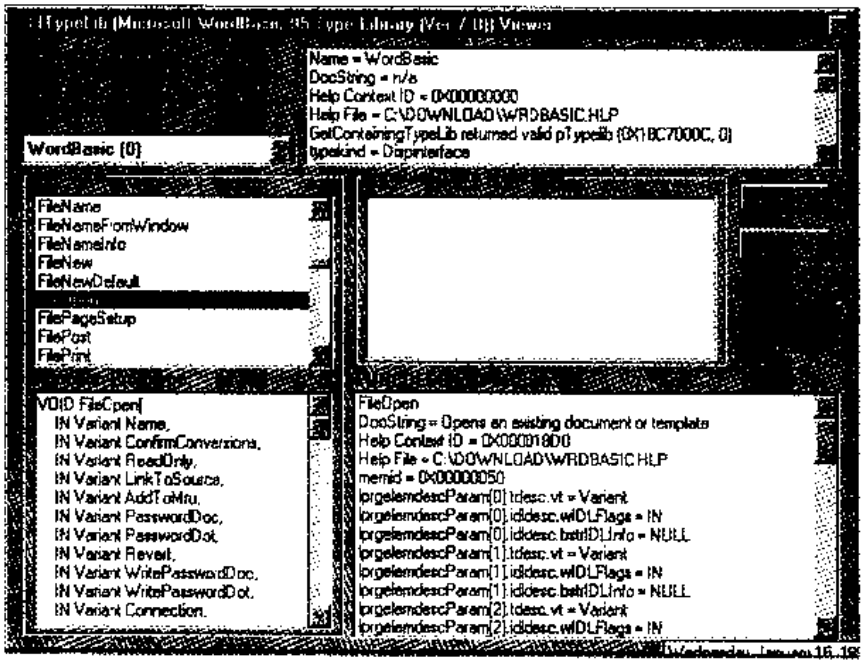


图 43.8 OLEview 和 OLE2view 允许用户检查 COM 对象中的方法

一般来说，对于基本 Java 数据类型如 int、short、double 等，有一个 put 方法。同样，对于由引用传送的参数(这种参数也可以返回一个值)，我们使用 put*XXX*Ref，如 putIntRef 或 putDoubleRef。我们也可以使用 get*XXX* 和 get*XXX*Ref 方法，检索存储在 Variant 对象中的值。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第七部分　高级 Java

第 44 章　Java 媒体框架
第 45 章　商业及 Java Wallet
第 46 章　数据结构及 Java 工具
第 47 章　java.lang
第 48 章　反射
第 49 章　使用其它语言扩展 Java
第 50 章　Java 与 C(++)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 44 章　Java 媒体框架

本章内容：

44.1　什么是 Java 媒体框架
44.2　创建媒体播放器
44.3　播放器的状态
44.4　向播放器添加控件
44.5　程序控制播放器
44.6　连接多个播放器
44.7　创建自己的媒体流
44.8　一个大型应用程序

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

44.1 什么是 Java 媒体框架

在 Java 的前两个版本中，程序员在使用 Java 时印象最深的是他们想要处理复杂的媒体类型如声音和视频数据的时候。的确 Java 当时支持一种音频类型(au)，但对复杂的类型如 MIDI 或 WAV 处理则是不可能的。另外，视频和动画也只能通过创建类似滑动图片的方式完成，不经过大量的工作，要处理 AVI 或 MOV 文件也是不可能的，而且也很不实用。随着 Java 媒体框(Media Framework)的出现，Java 已经进入了多媒体时代。Java 媒体框架提供了处理所有媒体类型的手段。它是一个将高级媒体格式如视频和音频集成到 Java 的 API。媒体框架中有多个组件，包括媒体播放器、媒体捕获以及会议组件。在这些课题中最关键的一点是如何提供一种计时机制，以确定什么时候播放媒体的下一部分。换句话说，寻找一种保持视频流的播放速度与其伴音流的播放速度相同的机制非常重要。否则，我们就会最终看到画面中人的嘴唇在动时，播放出的声音却是另外一个人的声音这种不一致的结果。

44.2 创建媒体播放器

为了理解 Java 媒体框架，最好是直接进入正题，通过创建一个小程序，来使用媒体播放器。将一个媒体播放器放在一个小程序中包括几个基本的步骤：

1. 为媒体文件创建一个 URL。
2. 为媒体创建一个播放器。
3. 通知播放器进行预提取。
4. 添加播放器到小程序。
5. 启动播放器。

第一步要做的事情我们在本书中已经做过数十次了。以下的步骤是创建媒体播放器本身所要求的。要完成这些步骤，我们要使用 Manager 类。Manager 类实际上是获得时间基和播放器的中心。我们需要将媒体文件所处位置的 URL 提供给管理器。

首要的任务是为媒体文件建立一个 URL，并创建一个播放器。在 BasicPlayer 类中，这些事情发生在 init( )方法中，如程序清单 44.1 所示。

[illegible] 与之相关的 URL

```
try {
        mediaURL = new URL(getDocumentBase(), mediaFile);
        player = Manager.createPlayer(mediaURL);
    }
    catch (IOException e) {
        System.err.println("URL for " + mediaFile + " is invalid");
    }
    catch(NoPlayerException e) {
        System.err.println("Could not find a player for " + mediaURL);
            }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

到现在我们应该熟悉程序清单 44.1 中第二行创建 URL 代码的方法了。但是仅为了复习，我们再简单重复一下。getDocumentBase( )方法返回一个 URL，该 URL 为小程序原始加载的位置，使用 mediaFile 表示相关的 URL，该 URL 为媒体文件所处的位置，而 mediaURL 最终指出完整正确的媒体文件的 URL。

程序清单中的下一行实际上是我们所感兴趣的。它请求管理器返回一个播放器，播放器知道如何处理我们所提供的 URL。

注意，此处我们需要捕捉两种异常。第一种是，如果 URL 非法，则 URL 构造器将抛出一个 IOException 异常。由于我们正在使用小程序的 documentBase，对这个特殊的例子发生这种异常是极不可能的。但是我们仍需要捕捉这种异常。

第二种异常是 NoPlayerException 异常。这是来自于媒体包中的一种新异常。当管理器得知没有为 URL 所指示的媒体类型所设计的播放器时，便产生这种特殊的异常。例如，如果我们指向一个 .wav 文件，而不存在 .wav 文件的播放器时，NoPlayerException 异常便产生。

44.2.1　预提取媒体

媒体处理过程中的下一步是预提取媒体。预提取引起两件事情发生。第一件为播放器要经过一个称为实现(realization)的过程。然后便开始卸载媒体文件，以便将其中的一部分放在高速缓存中。这将减少播放器在实际开始播放媒体之前的反应时间。在本章稍后，我们还要进一步讨论播放器可经过的各种状态，如预提取及实现。现在，认识到我们需要让播放器进行预提取就足够了。如程序清单 44.2 所示，在我们最基本的播放器中，使用 start( )方法进行预提取。

程序清单 44.2　在 start 方法中，我们预提取了我们将要播放的媒体

```
public void start() {
  if (player != null)
    //prefetch starts the player.
    //Prefetch returns immediately, just like getImage
    player.prefetch();
}
```

正如所看到的那样，对于预提取媒体我们需要做的就是调用播放器上的 prefetch 方法。

44.2.2　添加播放器到应用程序

添加播放器到应用程序实际上有一点小窍门。播放器本身并不是一个 AWT 组件。所以我们并不添加播放器本身，而是其可视的图像表示组件。为了得到可视组件，播放器有一个方法称为 getVisualComponent()。但在我们能够获得可视组件之前，要有一个捕捉，必须首先实现播放器。同样，我们在稍后会详细谈论这一点，但现在要认识到，播放器在被实现之前仍为无效，正如调用 getImage( ) 后一个图像仍未生效一样，这一点很重要。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

那么我们是如何知道组件是否已经被实现了呢？播放器的 getState 方法返回播放器的当前状态。如果需要，我们可以连续不断地检查其状态，当播放器的状态最终被实现之后，我们就可以请求可视组件了。

幸运的是 Java 提供了多种有效的方法。我们可以回忆一下，Image 有一个称为 ImageObserver 的接口，媒体 API 有一个类似的方法称为 ControllerListener。ControllerListener 有一个方法为 controllerUpdate(ControllerEvent)。我们可以使用 controllerUpdate 方法来检查是否已经预提取媒体。如程序清单 44.3 所示。

```
public synchronized void controllerUpdate(ControllerEvent event)
{
    if (event instanceof RealizeCompleteEvent)
    {
        // Once the player has been realized add the
        //visual component to the screen
        if ((visualComponent = player.getVisualComponent()) != null)
            add("Center", visualComponent);

        // draw the component
        validate();
    }
}
```

在控制器的状态每次改变后调用 controllerUpdate()方法。在本章的稍后，我们会看到，播放器本身就是一个控制器，它有许多不同的状态变化。

为了区分这些变化，可以产生许多事件。在此处的例子中，我们所需查找的是 RealizeCompleteEvent 事件。要确定所接收到的事件是哪种事件，我们使用 instanceof 操作符来判断当前事件是否是 RealizeCompleteEvent 类。如果是，那么我们知道播放器已经被实现，现在可以从播放器请求可视组件并将其添加到应用程序中。

getVisualComponent( )方法也有可能返回 null。当不存在指定媒体的可视组件时便返回 null。例如，音频文件播放器有可能不需要使用可视组件。显然，这意味着在添加可视组件到小程序之前我们需要测试一下 null 条件。

另外，为了确保组件在小程序中确实已经被描述，我们需要确认小程序已生效。validate()方法可使任何容器包括小程序，确认所添加的所有组件确实显示在屏幕上。

44.2.3 以侦听器注册小程序

在可以实际使用新的 controllerUpdate()方法之前，我们需要后退一步。我们或许奇怪，播放器是如何知道这种变化，去通知应用程序的呢？答案是它的确不知道。为了使播放器调用 controllerUpdate()，我们必须首先使用播放器注册应用程序。对传统的事件侦听器，这是由 addControllerListener()方法完成的。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

与所有 java.awt.event 侦听器一样，当一个组件作为侦听器被注册后，每次发生事件时都调用侦听器的方法（此例中为 controllerUpdate()）。对于我们现在的情况，我们将添加 addControllerListener 代码到小程序的 init( )方法中。完整的 init( )方法如程序清单 44.4 所示。

```
public void init() {
    String mediaFile = null;
    URL mediaURL = null;

    setLayout(new BorderLayout());

    if ((mediaFile = getParameter("Mile")) == null) {
      System.err.print ("Media File not present.");
      System.err.println("Required parameter is 'file'");
    }
    else {
      try {
        mediaURL = new URL(getDocumentBase(), mediaFile);

        player = Manager.createPlayer(mediaURL);

        if(player != null) {
          //tell the player to add this applet as a listener
          player.addControllerListener(this);
        }
        else
          System.err.println("failed to creat player for" + mediaURL);
      }
      catch (IOException e) {
        System.err.println("URL for " + mediaFile + " is invalid");
      }
      catch(NoPlayerException e) {
        System.err.println("Could not find a player for" + mediaURL);
      }
    }
  }
```

44.2.4　启动播放器

下一步就是通知播放器开始播放。正如请求可视组件一样，播放器在未达到某种状态之前是不能启动播放的。即播放器必须管理并完成我们前面所谈到的预提取。

从技术上讲，只要实现了播放器，我们就可以启动它了。但是，这样做并不明智，因为很有

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

可能媒体在播放时是不连贯的。媒体的预提取可使得播放器将某些媒体处理为流线型，以便它能够平滑地播放完整的内容。

为了确定预提取什么时候完成，我们可以查找一个 PrefetchCompletedEvent 事件。当这一事件发生后，预提取完成。为此我们将程序清单 44.5 中所给出的代码添加到 controllerUpdate 方法中。

程序清单 44.5 当预提取完成后启动播放器

```
if (event instanceof PrefetchCompleteEvent) {
  // start the player once it's been prefetched
  player.start();
}
```

44.2.5 清除及停止播放器

作为一个好的公共程序员，我们一定要使用小程序的 stop()方法自己进行清除。回想一下当浏览器离开当前 Web 页面时要调用 stop()方法。当浏览器离开页面时，我们也应当停止播放媒体。

停止播放器很简单，我们只要对播放器使用 stop()方法即可。但是还有另外一件事情需要我们去做——将媒体从内存中删去。可以想象，保持 2 到 3 兆已经不再需要的视频信息在内存中，是一种很大的浪费，并且将最终使整个系统性能降低。幸运的是播放器有完成这一任务的方法：deallocate( )。只要已经明确不再使用媒体信息，我们应该通知播放器解除媒体内存分配，以便能够进行垃圾回收。

使用播放器的 stop()和 deallocate()方法，我们可以创建小程序的 stop()方法。程序清单 44.6 给出了完整的 stop 方法。

程序清单 [illegible]

```
public void stop()
{
    if (player != null)
    {
        // Stop media playback
        player.stop();
        //release resources for the media
        player.deallocate();
    }
}
```

44.2.6 综合

现在，我们将刚刚学习的东西综合在一起放到一个完整的播放器中。程序清单 44.7 给出

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

完整的播放器样本。

```
import java.applet.*;
import java.awt.*;
import java.net.*;
import java.io.*;
import javax.media.*;

/**
 * A basic media player Applet
 */
public class BasicPlayer extends Applet implements ControllerListener {
  // the media player
  Player player = null;

  // Component where video will appear
  Component visualComponent = null;
  /**
   * Read the applet file parameter and create the media player.
   */
  public void init() {
    String mediaFile = null;
    URL mediaURL = null;

    setLayout(new BorderLayout());

    if ((mediaFile = getParameter("file")) == null) {
      System.err.print("Media File not present.");
      System.err.println("Required parameter is 'file'");
    }
    else {
      try {
        mediaURL = new URL(getDocumentBase(), mediaFile);

        player = Manager.createPlayer(mediaURL);

        if(player != null) {
          //tell the player to add this applet as a listener
          player.addControllerListener(this);
        }
        else
          System.err.println("failed to creat player for" + mediaURL);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
        catch (IOException e)
          System.err.println("URL for " + mediaFile + " is invalid");
        }
        catch(NoPlayerException e) {
          System.err.println("Could not find a player for" + mediaURL);
        }
      }
    }

    public void start() {
      if (player != null)
        //prefetch starts the player.
        //Prefetch returns immediately, just like getImage
        player.prefetch();
    }


    public void stop() {
      if (player != null) {
        // Stop media playback
        player.stop();
        //release resources for the media
        player.deallocate();
      }
    }


    //------ Controller interface method ----------

    /**
     * Whenever there is a media event,
     * the controllerUpdate method is called
     * for all the Player's listeners
     */
    public synchronized void controllerUpdate(ControllerEvent event) {
      if (event instanceof RealizeCompleteEvent) {
        // Once the player has been realized add
        // the visual component to the screen
        if ((visualComponent = player.getVisualComponent()) != null)
          add("Center", visualComponent);

        // draw the component
        validate();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }

        else if (event instanceof PrefetchCompleteEvent) {
          // start the player once it's been prefetched
          player.start();
        }
      }
    }
```

44.2.7　编译 BasicPlayer

在 JDK　1.2 下编译 BasicPlayer，只要使用 javac 编译器即可。但是，在 JDK　1.1 下也可以对 BasicPlayer 进行编译。为此，我们需要首先从软件商那里得到针对于我们的特殊计算机的 JMF 类的一个拷贝。对 Windows 95/98、Windows NT 和 Solaris 用户，可以从 Sun Microsystems 公司的如下地址获得一组类：

```
http://java.sun.com/products/java-media/jmf/1.0/
```

从这些站点下载并安装这些文件后，我们需要跟踪一下媒体类。对 Sun 的安装，我们所需要的文件为 jmf.jar，并安装到插件目录下(如果使用 Netscape 的 Navigator 的话)。

为了编译该程序，我们需要将 jmf.jar 文件包含在类路径中，然后再运行 javac 程序。

44.2.8　运行 BasicPlayer

在运行 BasicPlayer 之前，我们必须使用适当的文件创建一个 HTML 文件。在本例中，我们将使用包含在 JMF 类中的样本 MPEG 文件。我们的 HTML 文件应如程序清单 44.8 所示。

```
<html>
<body>
<applet code="BasicPlayer" width="320" height="300">
<param name="FILE" value="vample2.mpg">
Sorry, your browser does not support Java(TM) Applets.
</applet>
</body>
</html>
```

现在，运行 BasicPlayer 时我们有两种选择。第一种选择是，我们可以使用带有最后的 JDK 1.1 修补内容的 Netscape 4.03 或 Internet Explorer 4.0，如果我们已经正确安装了来自软件商的媒体文件，则现在可以打开 BasicPlayer.html 文件了。

第二种选择是，使用 JDK 中包含的 Appletviewer 程序。为了使用 Appletviewer，首先应确认在类路径中仍然包含着 media.zip 文件，然后像往常一样运行该程序：

```
Appletviewer BasicPlayer.html
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果一切都按计划进行，我们应该最终看到并听到宣传 Intel 的视频信息开始播放。

44.3 播放器的状态

现在我们已经完成了最基本的播放器的创建工作，该回过头来看看播放器在执行过程中都有哪些阶段或状态。图 44.1 给出了在一般操作过程中播放器所经历的各种状态。

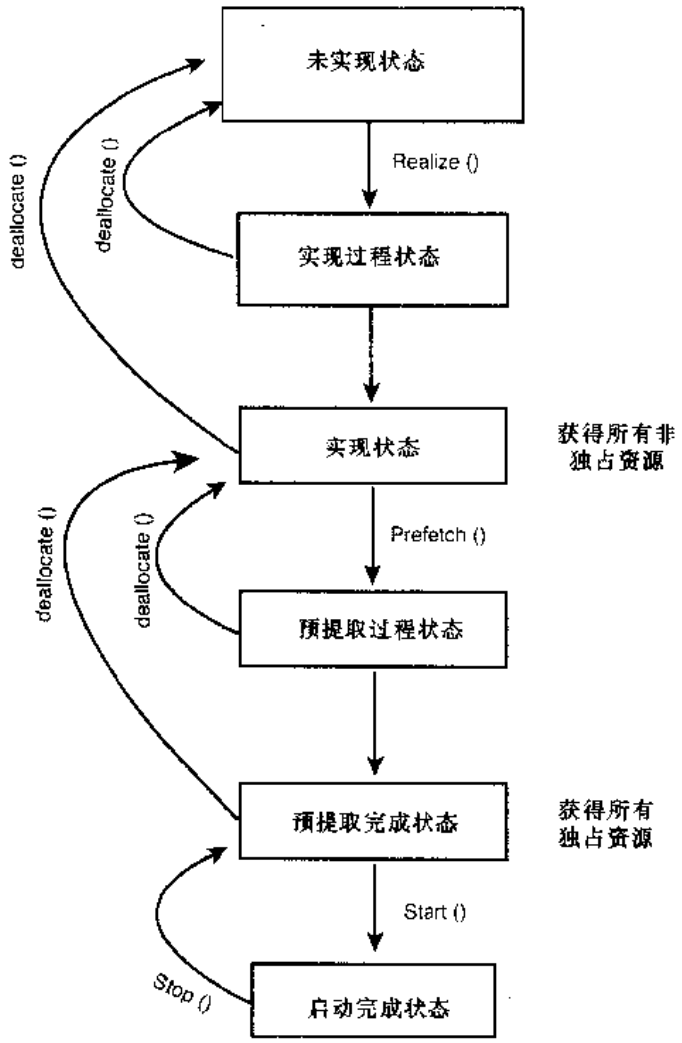


图 44.1 播放器在启动之前要经历许多种状态

从图 44.1 中可以看出，在每个方法调用之前我们需要跟踪播放器状态的原因。播放器的各种状态分别介绍如下：

未实现状态(Unrealized)：当我们最初创建播放器后，播放器处于未实现状态。在这种状态下，除了媒体的 URL 之外，播放器不能识别关于媒体的任务其它东西。为了使播放器进入下一阶段，我们可以使用 realize()方法。

> **注意**：如果播放器尚未被实现，prefetch()方法将自动引起 realize()方法的执行。所以，如果愿意我们可以像在 Basic Player 中所做的那样，跳过 realize()方法，直接进入 prefetch()方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

实现过程状态(Realizing):在实现过程状态中,播放器决定特定媒体所需的资源。例如,为了播放一个 FLIC 文件,它可能需要一个再现引擎。在实现过程状态中播放器获得所有那些非独占的资源。非独占资源指的是可以被多个播放器共享的那些资源。独占资源在稍后的预提取过程(prefetching)状态中获取。

实现状态(Realized):播放器进入实现状态时,发出 RealizeCompleteEvent 事件。在实现状态中,播放器知道了再现媒体所需的所有资源。另外,播放器对媒体本身也有充分的了解,可以给出媒体的可视组件。实现后的播放器与它所需的所有资源有一个"连接",但是它并没有真正拥有任何禁止其它播放器启动的资源。这些资源称为稀有资源,因为它们不能被系统中不同程序所共享。

预提取过程状态(prefetching):为使播放器进入预提取过程状态,我们可以使用 prefetch()方法。在预提取过程状态中,播放器预先装入准备播放的部分媒体。它也获得那些在实现过程状态中无法获取的稀有资源。

> **注意**:如果尚未提取媒体,start()方法将引起播放器调用 prefetch()方法。然而与 prefetch()不同,start()方法不能在未实现的播放器上调用。
>
> 还应注意,如果媒体位置发生变化,或媒体时间基改变并且需要卸载和缓冲存储额外的一些媒体数据,必须重新进入预提取过程状态。

预提取完成状态(Prefetched):当进入预提取完成状态后,播放器发出一个 PrefetchCompleteEvent 事件。当播放器处于预提取完成状态时,播放器便准备好了。它已经拥有了所有需要的资源,并且已经有了足够的媒体信息,可以开始播放了。简言之,播放器可以启动了。

启动完成状态(Started):正如我们在 BasicPlayer 中所看到的。要使播放器启动需要调用 start()方法。当播放器启动后,进入启动完成状态。当播放器启动后,其时钟开始运转,其媒体播放时间也已设置。如果播放器不是正在等待一个特殊的开始播放时间,则将立刻开始播放,否则,它将等待适当的时间,然后开始播放。

44.4　向播放器添加控件

现在我们在小程序中已经有了一个播放器,如果想使用户可以控制播放器的运行,我们将如何处理呢? Media API 已经提前为我们考虑到了。显然我们并不想自己为每种类型的播放器设计所需要的控件,使用 getControlPanelComponent()方法,播放器提供了一套控件。

与 getVisualComponent 方法一样,getControlPanelComponent()方法也不能在播放器被实现之前使用。如果回头看看前面的一节,我们将知道这是为什么了。在播放器被实现之前,它实际上并不知道将要播放的媒体类型。

程序清单 44.9 给出了控制面板的 BasicPlayer。图 44.2 给出了运行 BasicPlayer 时的状况。

程序清单 44.9　在 controlerUpdate 方法中添加控制面板是很简单的事情

```
import java.applet.*;
import java.awt.*;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.net.*;
import java.io.*;
import javax.media.*;

/**
 * A basic media player Applet
 */
public class BasicPlayer extends Applet implements ControllerListener {
  Player player = null;                    // the media player
  Component visualComponent = null;        // Component where video will appear
  Component controlComponent = null;
  /**
   * Read the applet file parameter and create the media player.
   */
  public void init() {
    String mediaFile = null;
    URL mediaURL = null;

    setLayout(new BorderLayout());

    if ((mediaFile = getParameter("file")) == null) {
      System.err.println("Media File not present. Required parameter is 'file'");
    }
    else {
      try {
        mediaURL = new URL(getDocumentBase(), mediaFile);

        player = Manager.createPlayer(mediaURL);

        if(player != null) {
          //tell the player to add this applet as a listener
          player.addControllerListener(this);
        }
        else
          System.err.println("failed to creat player for" + mediaURL);
      }
      catch (IOException e) {
        System.err.println("URL for " + mediaFile + " is invalid");
      }
      catch(NoPlayerException e) {
        System.err.println("Could not find a player for" + mediaURL);
      }
    }
  }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public void start() {
  if (player != null)
    //prefetch starts the player.
    //Prefetch returns immediately, just like getImage
    player.prefetch();
}


public void stop() {
  if (player != null) {
    // Stop media playback
    player.stop();
    //release resources for the media
    player.deallocate();
  }
}


//------ Controller interface method ----------

/**
 * Whenever there is a media event, the
 * controllerUpdate method is called
 * for all the Player's listeners
 */
public synchronized void controllerUpdate(ControllerEvent event) {
  if (event instanceof RealizeCompleteEvent) {
    // Once the player has been realized add
    // the visual component to the screen
    if ((visualComponent = player.getVisualComponent()) != null)
      add("Center", visualComponent);
    if ((controlComponent = player.getControlPanelComponent()) != null)
      if(visualComponent != null)
        add("South",controlComponent);
      else
        add("Center",controlComponent);


    // draw the components
    validate();
  }

  else if (event instanceof PrefetchCompleteEvent) {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        // start the player once it's been prefetched
        player.start();
      }
    }
  }
```

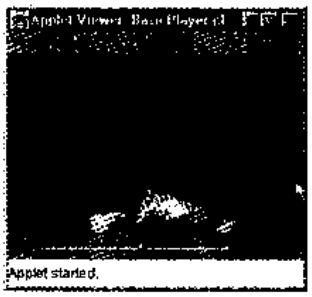


图 44.2 播放器有一个缺省的控制面板

44.5 程序控制播放器

除了使用控制面板以外,我们还可以在播放器中使用方法控制程序的运行。我们需要这样做有几方面的原因。原因之一可能是我们需要使用按钮,允许播放者控制播放器,而我们又不希望使用控制面板。另一个原因则是由于程序中其它一些条件的原因,我们可能需要改变播放器的播放方式。

44.5.1 启动播放器

正如在 BasicPlayer 实例中已经看到的那样,播放器有一个基本的方法 start(),该方法通知播放器开始播放。但是,还有一个更基本的方法,它允许我们启动一个播放器,并指定实际开始显示媒体信息的时刻。syncStart( )方法是实际引发播放器启动的方法。

首先,让我们简单回顾一下 start()方法完成一些什么工作。根据播放器的状态,start()方法将在其控制下顺序地首先为所有播放器调用 prefetch ()(本章稍后将讨论控制多个播放器的情况),并将播放器带入到预提取完成状态,然后调用 syncStart()。

44.5.2 设置媒体时间

我们常常喜欢重复观看一段影片,这与不全盘播放 CD 而只播放其中的曲目二和五没有什么不同。幸运的是媒体框架使用 setMediaTime()方法提供了这种功能。

setMediaTime()使用了一个长型数作为其参数,该数字以毫微秒为单位表示时间,与我们所熟悉的大多数时间增量不同(例如当休眠一个线程时),我们此处不是使用毫秒,而使用的是更精确的毫微秒。

44.5.3 改变播放速度

读者是否曾经希望提高或降低影片的播放速度呢? 如何回退播放呢? 播放器有一个专门用于此目的的方法。该方法为 setRate(),它允许用户改变媒体的播放速度。但是,使用 se-

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

tRate()方法只能是在播放器开始播放之前。当已经开始播放时若调用 setRate()方法，会产生一个异常。setRate()方法也可能不允许实际设置媒体播放速度。此时，能够保证的唯一一件事情是播放器将以 1.0 的速度播放媒体。setRate()方法返回实际使用的速度。

幸运的是，AVI 播放器支持以许多种不同的速度播放文件，所以我们可以向 BasicPlayer 添加改变速度的功能。如程序清单 44.10 所示，修改 controllerUpdate 方法。

```
public synchronized void controllerUpdate(ControllerEvent event)
{
if (event instanceof RealizeCompleteEvent)
{
    // Once the player has been realized add the visual component to the screen
    if ((visualComponent = player.getVisualComponent()) ! = null)
        add("Center", visualComponent);
    if ((controlComponent = player.getControlPanelComponent()) ! = null)
        if(visualComponent ! = null)
            add("South",controlComponent);
        else
            add("Center",controlComponent);


    // draw the components
        validate();
}

else if (event instanceof PrefetchCompleteEvent)
{
    System.out.println("prefetching:" + (new Date()));
    player.setRate((float)2.0);
    // start the player once it's been prefetched
    player.start();
}
}
```

44.5.4　改变音量

改变播放器音量的功能非常重要。为改变音量，JMF 提供了一个完整的类 GainControl。改变音量的第一步是从播放器获得 GainControl。为此，我们可以调用 getGainControl()方法。

有些播放器不具有音频组件，对这些播放器 getGainControl 方法返回 null 值。但对其它播放器，当获得 GainControl 后，我们就可以使用多种方式改变音量了。

对于音频可能最明显的就是彻底关闭它，或使其静音。对此，GainControl 类正好有一个

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用于此目的的方法 mute()。

除了完全静音之外，控制播放器音量还有两种截然不同的方式。第一种是基于一个刻度仪调整音量。刻度仪是 0 到 1.0 之间的一个任意数。当音量设为 1.0 时基本上达到最大音量，当设置为 0 时，几乎与静音设置一样。

为了改变 GainControl 音量刻度，我们可以使用 setLevel()方法。与之相对还有一个方法 getLevel()方法，用来检索 GainControl 的刻度。切记只有 0 到 1.0 之间的数为合法设置。

然而，刻度机制也存在问题，它不能使作为程序员的我们知道媒体实际播放时音量究竟有多大。一般地，音响工程师处理声音时使用分贝这一术语。刻度机制只是告诉我们，如果将刻度设置为 0.5，其音量大小只是设为 1.0 时的一半。幸好，媒体 API 还提供了另外一种设置音量分贝等级的手段，但是，这种技术实际上只是对媒体播放器具体情况的一种推测。它并不能解决像外部音箱的音量这样的问题，但它确实对我们自己的系统作出了一个合理的推测。

44.5.5 媒体播放器尺寸调整

媒体框架具有允许我们调整媒体组件尺寸的功能。显然，我们希望像其它组件一样，能够将媒体播放器的可视组件添加到一个面板上。为了使 LayoutManager 有效，必须能够调整媒体播放器的组件尺寸。

为了看到这一点，除了修改 HTML 文件中 BasicPlayer 的 WIDTH 和 HEIGHT 参数外，还要做一些其它工作。由于边界设计根据媒体尺寸自动地调整可视组件的尺寸，如果我们缩小小程序中的尺寸，将会看到媒体组件自动重新调整。请试用一下程序清单 44.11 中的 HTML 文件。图 44.3 给出了现在 BasicPlayer 的外形。

程序清单 44.11 BasicPlayer2.html——更小 width 和 height 参数的 HTML 文件

```
<html>
<body>
        <applet code="BasicPlayer" WIDTH="160" HEIGHT="150">
        <param name="FILE" value="ExampleVideo.avi">
        Sorry, your browser does not support Java(TM) Applets.
        </applet>
</body>
</html>
```



图 44.3 媒体框架自动调整到小程序指定的尺寸

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

44.5.6　添加进展条

我们知道，在 Internet 上卸载一个数兆字节的文件时需要很长时间。事实上，在卸载一个文件时会有一种总是不完的感觉。如果卸载占用的时间太长，用户甚至有可能以为整个系统已经挂起。

然而，如果有某种东西处于活动状态，用户就会知道系统并没有挂起。文件的卸载并不会少占用时间，但用户会觉得稍微可以忍受一些。建立某种活动的东西的完整方法是使用一个进展条。

回忆一下，每从 Internet 上下载一个图像的一部分，ImageObserver 都会得到一个通知。同样，媒体播放器也产生一个 CachingControlEvent 事件，在此之下是 CachingControl。大多数播放器都有一个控制，其唯一责任是掌握下载媒体文件的状态。更幸运的是，CachingControl 有一个进展条组件。与播放器所具有的可视组件和控制组件组件很相似，进展条组件可以添加到任何一个容器上。

使用进展条可大大增加用户的编程经验。程序清单 44.12 给出了添加进展条的 BasicPlayer。

```
import java.applet.*;
import java.awt.*;
import java.net.*;
import javax.media.*;
import java.util.Date;

/**
 * A basic media player Applet
 */
public class ProgressPlayer extends Applet implements ControllerListener
{
    Player player = null;                    // the media player
    Component visualComponent = null;        // Component where video will appear
    Component controlComponent = null;
    Component progressBar = null;
    /**
     * Read the applet file parameter and create the media player.
     */
    public void init()
    {
        System.out.println("init:" + (new Date()));
        String mediaFile = null;
        URL mediaURL = null;

        setLayout(new BorderLayout());
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        if ((mediaFile = getParameter("file")) == null)
        {
            System.err.println("Media File not present. Required parameter is 'file'");
        }
        else
        {
            try
            {
                mediaURL = new URL(getDocumentBase(), mediaFile);
                player = Manager.createPlayer(mediaURL);

                if(player != null)
                {
                    //tell the player to add this applet as a listener
                    player.addControllerListener(this);
                }
                else
                  System.err.println("failed to creat player for" + mediaURL);
            }
            catch (MalformedURLException e)
            {
                System.err.println("URL for" + mediaFile + " is invalid");
            }
            catch(NoPlayerException e)
            {
                System.err.println("Could not find a player for" + mediaURL);
            }
        }
    }

    public void start()
    {
        if (player != null)
            //prefetch starts the player.
            //Prefetch returns immediately, just like getImage
            player.prefetch();
    }


    public void stop()
    {
        if (player != null)
        {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        // Stop media playback
        player.stop();
        //release resources for the media
        player.deallocate();
    }
}


//- - - - - - Controller interface method - - - - - - - - - -

/**
  * Whenever there is a media event, the controllerUpdate method is called
 * for all the Player's listeners
 */
public synchronized void controllerUpdate(ControllerEvent event)
{
    if (event instanceof RealizeCompleteEvent)
    {
        // Once the player has been realized add the visual component to the screen
        if ((visualComponent = player.getVisualComponent()) != null)
            add("Center", visualComponent);
        if ((controlComponent = player.getControlPanelComponent()) != null)
            if(visualComponent != null)
                add("South",controlComponent);
            else
                add("Center",controlComponent);


        // draw the components
            validate();
    }

    else if (event instanceof PrefetchCompleteEvent)
    {
        System.out.println("prefetching:" + (new Date()));
        // start the player once it's been prefetched
        player.start();
    }
    else if (event instanceof CachingControlEvent) {

        // Put a progress bar up when downloading starts,
        // take it down when downloading ends.
        CachingControlEvent cce = (CachingControlEvent) event;
        CachingControl      cc = cce.getCachingControl();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            long cc_progress        = cce.getContentProgress();
            long cc_length          = cc.getContentLength();

            if (progressBar == null) // Add the bar if not already there ...
                if ((progressBar = cc.getProgressBarComponent()) != null) {
                    add("North", progressBar);
                    validate();
                }
                if (progressBar != null) // Remove bar when finished downloading
                        if (cc_progress == cc_length) {
                            remove (progressBar);
                            progressBar = null;
                            validate();
                        }
                }
        }
    }
```

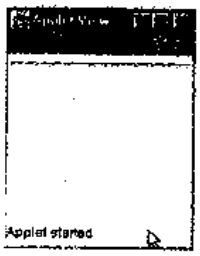


图 44.4 当播放器卸载媒体时进展条显示出其状态

44.6 连接多个播放器

当一个面板上有多个播放器时,很有可能希望使用一个控制器来控制它们的启动和停止。为此,我们需要能够连接这些控制器。

对这种需求播放器提供了一个方法:addController()。addController()方法的功能是通知播放器同时对其它播放器进行控制。对此要使用下述代码行:

```
masterPlayer.addController(slavePlayer);
```

slavePlayer 从属播放器由 masterPlayer 主播放器控制。然而,在可以控制其它播放器之前,从属播放器必须被实现。这样做的原因要回到播放器初建时它对自己的了解程度上来。我们知道,在一个播放器实现之前,它甚至不知道将要播放的媒体种类。所以它不知道将有一个什么样的控制器,也就不能将控制交给其它播放器。

为了获得另一个播放器的控制,两个播放器的时间基必须兼容。为确认它们的兼容性,主播放器要试着将其时间基传给从属播放器。如果这种操作失败,addController()将产生一个 IncompatibleTimeBaseException 异常。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

程序清单 44.13 是一个具有两个并行播放器的 BasicPlayer 例子，这些播放器由一个控制器控制。当播放器启动之后，我们只能看到一个控制面板。按播放按钮后，它们将同时启动。

```
import java.applet.*;
import java.awt.*;
import java.net.*;
import javax.media.*;
import java.util.Date;

/**
 * A basic media player Applet
 */
public class PrimaryPlayer extends Applet implements ControllerListener
{
    Player player = null;                    // the media player
    Player slavePlayer = null;
    Component visualComponent = null;   // Component where video will appear
    Component slaveVisualComponent = null;
    Component controlComponent = null;
    Component progressBar = null;
    /**
     * Read the applet file parameter and create the media player.
     */
    public void init()
    {
    String mediaFile = null;
        URL mediaURL = null;

        setLayout(new BorderLayout());

        if ((mediaFile = getParameter("file")) == null)
        {
            System.err.println("Media File not present. Required parameter is 'file'");
        }
        else
        {
            try
            {
                mediaURL = new URL(getDocumentBase(), mediaFile);
                player = Manager.createPlayer(mediaURL);
                slavePlayer = Manager.createPlayer(mediaURL);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            if(player ! = null)
            {
                //tell the player to add this applet as a listener
                player.addControllerListener(this);
            }
            else{
              System.err.println("failed to creat player for" + mediaURL);
        }

            if(slavePlayer ! = null)
            {
                //tell the player to add this applet as a listener
                slavePlayer.addControllerListener(this);
            }
            else{
              System.err.println("failed to creat player for" + mediaURL);
            }

        }
        catch (MalformedURLException e)
        {
            System.err.println("URL for " + mediaFile + " is invalid");
        }
        catch(NoPlayerException e)
        {
            System.err.println("Could not find a player for" + mediaURL);
        }

}
}

public void start()
{
    if (player ! = null)
        //prefetch starts the player.
        //Prefetch returns immediately, just like getImage
        player.prefetch();
    if (slavePlayer ! = null)
        slavePlayer.prefetch();
}


public void stop()
{
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    if (player ! = null)
    {
        // Stop media playback
        player.stop();
        //release resources for the media
        player.deallocate();
    }

//- - - - - - Controller interface method - - - - - - - - - - -

/* *
  * Whenever there is a media event, the controllerUpdate method is called
 * for all the Player's listeners
 */
public synchronized void controllerUpdate(ControllerEvent event)
{
    if (event instanceof RealizeCompleteEvent)
    {
        System.out.println("realized");
        // Once the player has been realized add the visual component to the screen
        if (event.getSource() = = slavePlayer){
            System.out.println("slave");
            if ((slaveVisualComponent = slavePlayer.getVisualComponent()) ! = null)
                add("West", slaveVisualComponent);
            try{
                player.addController (slavePlayer);
            }
            catch (IncompatibleTimeBaseException e)
            {
                System.err.println("Could not attach player" + e);
            }
            validate();

        }
        else{
        if ((visualComponent = player.getVisualComponent()) ! = null)
                add("East", visualComponent);
        if ((controlComponent = player.getControlPanelComponent()) ! = null)
                if(visualComponent ! = null)
                    add("South",controlComponent);
                else
                    add("Center",controlComponent);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            // draw the components
                validate();
            }

        }

        if (event instanceof PrefetchCompleteEvent){
        //player.setRate((float)2.0);
            // start the player once it's been prefetched
        //player.setMediaTime(20000);
            player.start();
        }
        else if (event instanceof CachingControlEvent) {

            // Put a progress bar up when downloading starts,
            // take it down when downloading ends.

            CachingControlEvent   e = (CachingControlEvent) event;
            CachingControl        cc = e.getCachingControl();
            long cc_progress         = e.getContentProgress();
            long cc_length           = cc.getContentLength();

            if (progressBar == null) // Add the bar if not already there ...
                if ((progressBar = cc.getProgressBarComponent()) != null) {
                    add("North", progressBar);
                    validate();
                }
            if (progressBar != null) // Remove bar when finished downloading
                if (cc_progress == cc_length) {
                    remove (progressBar);
                    progressBar = null;
                    validate();
                }
        }
    }
}
```

44.7 创建自己的媒体流

44.7.1 被动媒体流

我们知道，为了使数据到达客户端有多种方式。更重要的是，对检索数据有两个基本的子

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

集。一个基本的类型为被动(pull)媒体流,这种情况下,客户请求服务器向其发送信息,这正是 Web 浏览器的工作方式。我们退到一个 Web 站点,并单击一个 URL,浏览器请求服务器向其发送 Web 页面,然后继续其它工作。另外一种数据流称为主动(push)媒体流,我们稍后再讨论。主动媒体流的结果是服务器自动向客户发送信息,某种程序上就像是向客户广播信息。

被动媒体流是许多 Internet 协议普遍采用的方式。例如,Internet 上两个最流行的协议 HTTP 和 FTP 都是被动流的例子。我们已经看到过 HTTP 协议在 BasicPlayer 中是如何使用的。

下面,我们将探讨如何建立自己的被动流。

44.7.2　主动媒体流

最近由于 PointCost 和 Marimba 的出现,主动流也开始流行起来了。被动流的一个基本优点是可以保证客户百分之百地接收到服务器所发送的数据。因此,播放器不需要知道如何调整数据间的间隙,而这种情况在主动流的情况下很可能发生。主动播放器需要知道当遇到数据间隙时如何处理,并且必须能够处理。

44.8　一个大型应用程序

现在,我们已经学习了关于播放器的全部控制,我们可以将这些内容放在一起,创建一个自己定制的控制面板。我们将使用到所有已经学习过的方法。见程序清单 44.14。

```
import java.applet.*;
import java.awt.*;
import java.net.*;
import java.media.*;

/**
 * This is a Java Applet that demonstrates how to add your own
 * custom controls to the basic media player.
 */
public class ExtendedPlayer3 extends Applet implements ControllerListener
{
    Player player = null;                    // media player
    Component visualComponent = null;        // component in which the video is playing
    boolean running = false;                 // indicates if the applet
➥is currently running, because the user is on the page

    /**
     * The component of the media player that holds the gainControl reference.
     */
    GainControl gainControl = null;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    /**
     * Panel used to hold the custom controls in the Applets Layout Manager.
     */
    Panel controlPanel = null;

  /**
     * Runnable class used to monitor media progress and update the UI.
     */
    MediaProgressMonitor progressMonitor = null;

    /**
     * Buttons within the custom controls used to adjust Starting, Stopping,
     * Gain increase and Gain decrease.
     */
    Button startButton = null;
    Button stopButton = null;
    Button gainUpButton = null;
    Button gainDownButton = null;

     /**
     * Labels within the custom controls used to indicate adjustable media player
     * features.
     */
    Label controlLabel = null;
    Label gainLabel = null;
    Label muteLabel = null;
    Label mediaTimesLabel = null;

    /**
     * Checkbox within the custom controls used control the Mute feature of the
     * media player.
     */
    Checkbox muteCheckbox = null;

    /**
Textfield within the custom controls used to indicate
*  the current media player
     * position and total media file duration.
     */
    TextField timeText = null;

    /**
     * Read the applet file parameter and create the media player.
     */
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public void init()
{
    String mediaFile = null;      // input filename from Applet parameter
    URL mediaURL = null;          // base URL for the document containing the applet

    setLayout(new BorderLayout());

    /**
     * Get the media filename info.
     * The applet tag should contain the path to the
     * source media file, relative to the applet.
     */
    if ((mediaFile = getParameter("MediaFile")) == null)
    {
        System.err.println("Invalid media file parameter");
    }
    else
    {
        try
        {
            // Create an url from the file name and the url to the
            // document containing this applet.
            mediaURL = new URL(getDocumentBase(), mediaFile);

            // Create an instance of an appropriate media player for this media type.
            player = Manager.createPlayer(mediaURL);

            if(player != null)
            {
                // Add this applet as a listener for the media player events
                player.addControllerListener(this);

                // Create the duration monitor object:
                progressMonitor = new MediaProgressMonitor(this);
            }
            else
             System.err.println("Could not create player for" + mediaURL);
        }
        catch (MalformedURLException e)
        {
            System.err.println("Invalid media file URL!");
        }
        catch(NoPlayerException e)
        {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                System.err.println("Could not find a player to create for" + mediaURL);
            }
        }
    }

    /**
     * Start media file playback. This method is called the first time
     * that the Applet runs and every time the user re-enters the page.
     *
     * Call prefetch() to prepare to start the player. Prefetch returns
     * immediately, so this method does not call player.start(). The
     * controllerUpdate() method will call player.start() once the
     * player is Prefetched.
     */
    public synchronized void start()
    {
        if (player != null)
        {
            player.prefetch();
            running = true;
        }
    }

    /**
     * Stop media file playback and release resources before leaving
     * the page.
     */
    public synchronized void stop()
    {
        if (player != null)
        {
            progressMonitor.stop();
            player.stop();
            player.deallocate();
            running = false;
        }
    }

    /**
     * This controllerUpdate method must be defined in order to implement
     * a ControllerListener interface. This method will be called whenever
     * there is a media event.
     */
    public synchronized void controllerUpdate(ControllerEvent event)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
{
    // do nothing if player is set to null
    if (player == null)
        return;
    // When the player is Realized, get the visual component
    // and control component and add them to the Applet
    if (event instanceof RealizeCompleteEvent)
    {
        if ((visualComponent = player.getVisualComponent()) != null)
            add("Center", visualComponent);

        // Get pointer to the Gain Control of the media player.
        gainControl = player.getGainControl();

        // Create the custom control components.
        createCustomControls();
        if (visualComponent != null)
            add("South",controlPanel);
        else
            add("Center",controlPanel);

        // force the applet to draw the components
        validate();
    }

    // Once the player has Prefetched, start it
    else if (event instanceof PrefetchCompleteEvent)
    {
        if(running)
        {
            player.start();
            progressMonitor.start();
        }
    }

    // If we've reached the end of the media "rewind" it to the beginning.
    else if (event instanceof EndOfMediaEvent)
    {
        player.setMediaTime(0);
        if (running)
            player.start();
    }

    // A fatal player error has occurred
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        else if (event instanceof ControllerErrorEvent)
        {
            progressMonitor.stop();
            player = null;
            System.err.println("FATAL ERROR: " + ((ControllerErrorEvent)event).getMessage());
        }
    }

    /**
     * This method handles the AWT details required to display
     * player controls.
     */
    public void createCustomControls()
    {
        controlPanel = new Panel();

        GridBagLayout gridBag = new GridBagLayout();
        GridBagConstraints gridBagCon = new GridBagConstraints();

        controlPanel.setFont(new Font("Arial", Font.PLAIN, 14));
        controlPanel.setLayout(gridBag);

        // Create the first row of AWT control components: Label, Start Button, Stop Button.
        gridBagCon.fill = GridBagConstraints.BOTH;
        gridBagCon.weightx = 1.0;
        controlLabel = new Label("Controls:", Label.LEFT);
        makeControl(controlPanel, controlLabel, gridBag, gridBagCon);

        startButton = new Button("Start");
        makeControl(controlPanel, startButton, gridBag, gridBagCon);

        gridBagCon.gridwidth = GridBagConstraints.REMAINDER;
        stopButton = new Button("Stop");
        makeControl(controlPanel, stopButton, gridBag, gridBagCon);

        // Create the second row of AWT control components: Label, GainUp Button, GainDown Button.
        gridBagCon.weightx = 1.0;
        gridBagCon.gridwidth = 1;
        gainLabel = new Label("Gain:", Label.LEFT);
        makeControl(controlPanel, gainLabel, gridBag, gridBagCon);

        gainUpButton = new Button("Loud");
        makeControl(controlPanel, gainUpButton, gridBag, gridBagCon);
        gridBagCon.gridwidth = GridBagConstraints.REMAINDER;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        gainDownButton = new Button("Soft");
        makeControl(controlPanel, gainDownButton, gridBag, gridBagCon);

    // Create the third row of AWT control components: Label, Mute checkbox.
        gridBagCon.gridwidth = GridBagConstraints.RELATIVE;
        gridBagCon.weightx = 1.0;
        muteLabel = new Label("Mute:", Label.LEFT);
        makeControl(controlPanel, muteLabel, gridBag, gridBagCon);

        gridBagCon.gridwidth = GridBagConstraints.REMAINDER;
        muteCheckbox = new Checkbox(" ");
        makeControl(controlPanel, muteCheckbox, gridBag, gridBagCon);

        // Create the third row of AWT control components:Label, media time textbox.
        gridBagCon.gridwidth = 1;
        gridBagCon.weightx = 1.0;
        mediaTimesLabel = new Label("Current Time/Total Time:", Label.LEFT);
        makeControl(controlPanel, mediaTimesLabel, gridBag, gridBagCon);
        gridBagCon.gridwidth = GridBagConstraints.REMAINDER;
        timeText = new TextField("0.0//0.0", 12);
        timeText.setEditable(false);
        makeControl(controlPanel, timeText, gridBag, gridBagCon);
    }

    /**
     * This method adds a control to the custom control layout manager.
     */
    protected void makeControl(Container parentComp, Component
    newComponent, GridBagLayout gridbag, GridBagConstraints constraint)
    {
        gridbag.setConstraints(newComponent, constraint);
        parentComp.add(newComponent);
    }

    /**
     * This method captures all the events from the custom controls. This
     * is where each control calls the media player control methods.
     */
    public boolean action(Event evt, Object arg)
    {
        if (evt.target instanceof Button)
        {
            // Process the button event:
            if ("Start".equals(arg))
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            {
                player.start();
            }
            else if ("Stop".equals(arg))
            {
                player.stop();
            }
            else if ("Loud".equals(arg))
            {
                gainControl.setDB(2.0f);
            }
            else if ("Soft".equals(arg))
            {
                gainControl.setDB(-2.0f);
            }
            return(true);
        }
        else if (evt.target instanceof Checkbox)
        {
            // Set the player's Mute control based upon the checkbox state.
            if (muteCheckbox.getState() == true)
                gainControl.setMute(true);
            else
                gainControl.setMute(false);
            return(true);
        }
        else
            return(false);
    }
}

/**
 *      This class is used to continually monitor the progress
 *      of the playing media file. The thread wakes up every 50 millisec
 *      and passes the progress info to the player controls in the applet.
 */
class MediaProgressMonitor implements Runnable
{
    ExtendedPlayer3 m_Applet = null;
    Thread thread = null;
    boolean running;

    public MediaProgressMonitor(ExtendedPlayer3 applet)
    {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        m_Applet = applet;
    }

    /**
     * This method is called when the user starts the Applet or returns
     * from another page.
     */
    public synchronized void start()
    {
        thread = new Thread(this);
        thread.start();
        running = true;
    }

    /**
     * This method is called when the user stops the Applet or
     * leaves the page.
     */
    public synchronized void stop()
    {
        running = false;
    }

    /**
     * This method is called after the start method has executed.
     * Every 50 milliseconds, check the media player's progress
     * and forward the results to the player's control component.
     */
    public void run()
    {
        String mediaDuration = null;
        String mediaTime = null;
        char tmpChar;

        // Get the total time of the media file and store for use later.
        long duration = m_Applet.player.getDuration();
        mediaDuration = new String(String.valueOf(duration / (long) 1e08));
        tmpChar = mediaDuration.charAt(mediaDuration.length() - 1);
        mediaDuration = String.valueOf(duration / (long) 1e09) + "." + tmpChar;
        while (running)
        {
            // Update the media time text field.
            long currtime = m_Applet.player.getMediaTime();
            mediaTime = new String(String.valueOf(currtime / (long) 1e08));
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                tmpChar = mediaTime.charAt(mediaTime.length() - 1);
                mediaTime = String.valueOf(currtime / (long) 1e09) + "." + tmpChar;
                m_Applet.timeText.setText(mediaTime + "/" + mediaDuration);

                try
                {
                    thread.sleep(50);
                }
                catch (InterruptedException e)
                {
                    System.err.println(e);
                }
            }
        }
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 45 章　商业及 Java Wallet

本章内容：

45.1　JCC 的安全性支持

45.2　商业消息

45.3　创建盒式带

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

45.1 JCC的安全性支持

随着 Internet 上用户增多,越来越多的公司希望能在线销售其产品和服务。在线销售的最大问题之一是安全性。虽然 Java Security API 为安全地在 Internet 传送信用卡及银行信息提供了很强的加密功能,但是为了进行电子交易,仍需要一套普通的 API。Java Commerce Client(Java 商用客户,JCC)就是为了支持在线交易所需的一切特性而设计的。

JCC 包括一套标准的用户接口,一个客户与服务器之间传送消息的框架,以及一套处理在线交易各方面问题的 API。

JCC 的核心特性之一是盒式带(Cassette),它是一组执行特殊操作的类。我们可以将盒式带想象成为对象的更大一些的版本——它包括多个对象和一些其它信息。它"插接"到商业框架上,用来执行一些所需的功能。

> **注意**:Java Commerce Client 最初称为 Java Electronic Commerce Framework(Java 电子商务框架,JECF)。在许多地方我们仍会看到把它称为 JECF,包括 Sun 公司的文档中。

45.2 商业消息

在一个客户/服务器环境中,尽早明确所使用的协议非常重要,这样便于客户和服务器端开发工作的进行。JCC 定义了一种消息格式称为 Java Commerce Messages(Java 商业消息,JCM),它使得客户与服务器之间交互的定义变得很容易。

JCM 消息是人可识别的文本消息,包括多个 name = value 对,如:

```
buyer.billto.name = Mark Wutka
```

消息的内容会随着服务器的不同而不同。一般来说,消息应包括关于购买者、订单以及其它所有必须的信息。

JECF 还为 JCM 定义了一个 MIME 类型,以便电子邮件(e-mail)程序、Web 服务器、浏览器以及其它可识别 MIME 的应用程序可以处理 JCM 消息。MIME 类型为 application/x-java-commerce,JCM 的文件扩展名为 jcm。

一般情况下,由服务器决定执行一个交易所需发送的数据种类。它们有可能是一组每个服务器都期待的 name = value 对,也可能是随服务器类型而变化的服务器特定的其它扩展。

45.3 创建盒式带

盒式带(Cassette)是 JCC 的内核,它执行从用户获得信息、合法交易及发送信息到服务器等所需的所有操作。盒式带被分成不同的区,用来说明它所执行的操作类型。盒式带的类型有工具、协议、操作、服务及用户接口(UI)等类型。

工具盒式带表示交易中所使用的数据。一个典型的工具盒式带可能代表一个信用卡,包括信用卡的类型、号码及截止日期。另外,工具也可能代表一个银行帐户,包括银行的位置、帐户号以及帐户类型。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

协议盒式带是 JCC 与商业服务器之间的一种通信机制。我们可能在涉及网络的领域内已经遇到过“协议”这一术语，在网络领域内我们可以发现有文件传输协议(File Transfer Protocol，FTP)和超文本传输协议(Hypertext Transfer Protocol，HTTP)。在电子商务领域内，我们可以发现有一种称为 SET 的协议，该协议是安全电子交易的一个标准。虽然 SET 定义了客户与服务器之间的交互，协议盒式带实际上是将操作盒式带所需要的各种协议映射到通信协议上。例如，JCC 有一个为进行买入而定义的协议。一个针对 SET 协议的协议盒式带实现了买入协议和使用 SET 协议执行买入的接口。

操作盒式带代表用户需要执行的任务。一个很通用的操作盒式带是实现买入操作的操作盒式带，它可使用户完成一次买入。我们或许还需要一个卖出操作，使用户通过电子手段卖出东西。大多数情况下，卖出是在服务器上执行的，然而也有用户进行卖出的情况，如在股票交易中。

UI 盒式带表示执行各种操作的用户接口。例如，ATM UI 或许表示了在自动提款机中可以找到的所有操作，尽管由于不能在调制解调器上传送纸币，也不可能存储通过传真接收到的现金，这使得分发现金有些可笑。然而，我们可以通过帐户传输现金和填写借记卡。

服务盒式带是一个“帮助器”盒式带，它本身并不需要执行交易。一个通用的服务盒式带是信用卡的 Rolodex 或一个可视的股票交易机。服务盒式带也可以为其它的盒式带提供服务。例如，操作盒式带可能使用服务盒式带提供执行操作时的用户接口。

45.3.1　CassetteControl 类

每个盒式带中都有一个名为 CassetteControl 的类，其中包含了关于盒式带内容、所执行的功能种类、当前版本以及它可能依赖的其它盒式带等信息。当盒式带第一次安装到 JCC 环境时要使用到 CassetteControl 类，当检查其版本时也要使用到它。

在创建自己的 CassetteControl 类时，必须将其声明为 Cassette 的一个子类，不存在 CassetteControl 基类。Cassette 类加载器特别地寻找一个 CassetteControl 的类，并希望它是 Cassette 的子类。

在创建 CassetteControl 类对象时我们所需要的一个“帮助器”类是 CassetteIdentifier，它包括了盒式带的名字及版本信息。创建 CassetteIdentifier 可使用如下三种方式：

```
CassetteIdentifier()
CassetteIdentifier(String name)
CassetteIdentifier(String name,int majorVersion,int minorVersion)
```

如果使用 null (空)构造器创建空的盒式带标识，则必须使用 setName 、setMajorVersion 和 setMinorVersion 方法设置其名字和版本号：

```
public void setName(String name)
public void setMajorVersion(int majorVersion)
public void setMinorVersion(int minorVersion)
```

如果仅使用名字设置盒式带标识，则必须将版本信息以如下方式嵌入到名字中：

```
name _ majorVersion.minorVersion
```

例如，下面的两个盒式带标识构造器创建了相同的标识：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CassetteIdentifier("StockTrader",3,1)
CassetteIdentifier("StockTrader_3.1")
```

在盒式带控制对象中使用盒式带标识来提供盒式带的当前版本号,以及所依赖的其它盒式带的标识。

每个盒式带控制对象必须在 getCurrentVersionIdentifier 方法中返回当前的版本。下面的代码段创建了一个版本标识:

```
public CassetteIdentifier getCurrentVersionIdentifier( )
{
    return new CassetteIdentifier("StockTrader",3,1);
}
```

当然,事先创建一个盒式带标识,并在每次调用 getCurrentVersionIdentifier 时返回该标识会更有效。

当返回某个盒式带所依赖的盒式带时,也使用到了盒式带标识。getDependencyIdentifiers 方法返回一个盒式带标识数组,或在没有所依赖的盒式带时返回 null。例如,假设我们的盒式带依赖于名为 TradeOMatic 和 LeatherPortfolio 的盒式带,则我们的 getDependencyIdentifiers 方法应如下所示:

```
public CassetteIdentifier[] getDependencyIdentifiers()
{
    CassetteIdentifier dependencies[] = new CassetteIdentifier[2];

    dependencies[0] = new CassetteIdentifier("TradeOMatic",2,1);
    dependencies[1] = new CassetteIdentifier("LeatherPortfolio_2.4");

    return dependencies;
}
```

getJCMForLatestVersion 方法返回一个 URL 数组,它告诉 JCC 到何处去找更新的盒式带版本:

```
public URL[] getJCMForLatestVersion( )
```

如果另一个盒式带需要我们的盒式带的新版本,JCC 将自动检查这些 URL 并卸载所需的版本。

CassetteControl 中的安装(install)方法使用下列方法之一对其本身在 JCC 注册:

```
public final void registerInstrumentType(String instrumentType,String className)
public final void registerProtocol(String protocolName, String className)
public final void registerOperation(String operationName, String className)
public final void registerService(String serviceName, String className)
public final void registerWalletUI(String walletUIName, String description,String className)
```

对 StockTrader 操作的安装(install)方法应如下所示:

```
public void install( )
    throws CassetteInstallationException
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
{
    registerOperation("StockTrader","stocks.trader.cassettes.StockTrader");
```

当一个盒式带从系统中删去时，JCC 调用一个 uninstall 方法。如果我们的盒式带在卸载时不需要做其它的事情，那么只需创建一个空的方法即可，如下所示：

```
public void uninstall()
{
}
```

当一个盒式带启动时，要调用初始化(init)方法。当盒式带停止时，要调用关闭(shutdown)方法：

```
public void init()
public void shutdown()
```

然而，对于 CassetteControl 类，无需初始化(init)和关闭(shutdown)方法，只要使它们为空方法即可。

GetExpirationDate 方法返回盒式带的截止日期：

```
public Date getExpirationDate( )
```

当系统需要更新一个盒式带时，它首先询问盒式带是否允许更新。传送给 doUpdate 方法上次更新的日期，如果允许更新则该方法返回 true(真)，如果不应该进行更新则返回 false(假)：

```
public boolean doUpdate(Date lastUpdate)
```

45.3.2　工具盒式带类

在 JCC 范围内，工具代表的是数据源。通常，工具代表诸如信用卡、银行帐户或股票之类的东西。典型的情况下，一个正在访问在线商店的用户要决定进行一次买入，JCC 小程序会提供一组工具，这组工具也许看上去像是一个钱夹子，用户将选择其中的一种“工具”进行支付，如使用信用卡等。小程序将使用这种工具，以及一个操作盒式带和一个协议盒式带，将订单传送到在线商店。操作盒式带和协议盒式带将与信用工具进行通信，获得完成这次销售所需的所有信息——帐户号码、截止日期等。

工具实现了工具(Instrument)接口，工具接口是每个工具都支持的接口的一种高级描述。之所以是一种高级描述，是因为不同接口的变化非常大，不可能预先知道所有需要的方法并将其放置到接口中。此外，我们并不需要股票工具中也包含获得信用卡号码和截止日期的方法。相反，各种类型的工具有其独立的接口，来实现工具接口。

例如，JCC 带有一个称为 GenericCreditCard 的接口用来实现工具(Instrument)接口。正如我们所希望的那样，GenericCreditCard 接口中有查询帐户号码、截止日期、帐单地址及持卡人姓名等信息的方法。

工具接口包含了表示工具类型及一般功能的函数。例如 getDescription 方法返回关于工具的一个简单文本描述。

```
public String getDescription( )
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

而 getType 方法返回工具的类型，如“Visa”、“MasterCard”、“Amex ”或“Discover”：

```
public String getType()
```

getName 方法返回工具的名字，如“Sir ChargeAlot”：

```
public String getName()
```

getContext 方法返回一个描述工具上下文关系的字符串：

```
public String getContext()
```

目前的 JCC 文档给出了上下文关系的例子“pay”和“accumulate”。这些字符串在将来某一天有可能像所希望的那样被标准化，这样就可以避免可能出现的术语冲突。

getVisualRepresentation 方法返回一个表示工具的 AWT 组件：

```
public Component getVisualRepresentation(CommerceContext context,Dimension dim)
```

工具的可视表示主要考虑了市场上人们所喜欢的一种俏丽的表示方式。可视表示可以显示动画，甚至可以播放音乐片段。（对信用卡，作者推荐使用 Eagles 的“Trade It to the Limit”。）

getSimpleGraphic 方法返回一个使用于各种选择屏上的图象：

```
public Image getSimplerGaphics(CommerceContext context,Dimension dim)
```

简单图像用于显示用户用来从工具组中进行选择的容器。商业小程序可能会显示出一个包含有信用卡的钱夹子的图像。当我们选择了信用卡之后，小程序将对其余的事务使用可视表示的对象。

InstrumentAdministration 接口包含了关于创建和维护特定工具的方法。其中包括几个方法仅仅用来获取用于编辑工具的图像组件。例如，getNewInstrumentUI 方法返回一个用于创建新工具的 AWT 容器：

```
public Container getNewInstrumentUI(DataStore instStore,CommerceContext context, Dimension dim)
```

另一方面，getInstrumentEditUI 方法返回一个用于编辑现有工具的容器：

```
public  Container  getInstrumentEditUI(DataStore instStore,
    CommerceContext context, Dimension dim)
```

getInstrument 方法从数据库返回一个工具：

```
public Instument getInstrument(DataStore instStore)
```

DataStore 对象使用于 getInstrument 方法和 new/edit 用户接口，它提供了一种在数据库中存储和检索工具的机制。DataStore 实际上表示了一个数据库 blob（Binary Large Object，二进制大型对象），在这种数据库 blob 中工具的存储使用了 Java 系列化 API。DataStore 对象只有三个方法：

```
public boolean commit() throws IOException
public void setObject(Serializable obj) throws IOException
public Serializable getObject() throws IOException
```

setObject 方法将对象存储到一个数据存储区中，getObject 方法从数据存储区中检索一个

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

对象。而 commit 方法保存所做的数据存储修改，如果保存成功它返回 true(真)。

由于信用卡是目前所使用的支付方式中最流行的形式之一，JCC 包含了一种一般信用卡接口，该信用卡接口提供了在普遍使用的信用卡中能够找到的信息。这就为协议和操作盒式带的实现者提供了一个可以开始工作的基础，他们不必再等待，观察其它开发者是如何实现信用卡工具了。如果我们创建一个信用卡工具，它应实现 GenericCreditCard 接口。

GenericCreditCard 接口包含用来获取和设置信用卡上各种项目的 get 和 set 方法。get/set 方法如下所示：

```
public String getPAN( );//PAN = Primary Account Number(主帐户号码)
public void setPAN (String primaryAccountNumber);

public String getExpireDate( );
public void setExpireDate(String expireDate);

public String getCardholderName( );
public void setCardholderName(String name);

public AddressRecord getBillingAddress( );
public void setBillingAddress(AddressRecord address);
```

另外，如果当前的交易得到允许，则 accept 方法将返回 true(真)：

```
public boolean accept();
```

程序清单 45.1 给出了 Sun 公司的 JCC 包中的一个工具实例。

```
/* @(#)CCInstrument.java 1.21 11/07/97
*
* Copyright (c) 1996 Sun Microsystems, Inc. All Rights Reserved.
*
* Permission to use, copy, modify, and distribute this software
* and its documentation for NON-COMMERCIAL or COMMERCIAL purposes and
* without fee is hereby granted.
* Please refer to the file http://java.sun.com/copy_trademarks.html
* for further important copyright and trademark information and to
* http://java.sun.com/licensing.html for further important licensing
* information for the Java (tm) Technology.
*
* SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
* THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR
* ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
 *
 * THIS SOFTWARE IS NOT DESIGNED OR INTENDED FOR USE OR RESALE AS ON-LINE
 * CONTROL EQUIPMENT IN HAZARDOUS ENVIRONMENTS REQUIRING FAIL-SAFE
 * PERFORMANCE, SUCH AS IN THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT
 * NAVIGATION OR COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL, DIRECT LIFE
 * SUPPORT MACHINES, OR WEAPONS SYSTEMS, IN WHICH THE FAILURE OF THE
 * SOFTWARE COULD LEAD DIRECTLY TO DEATH, PERSONAL INJURY, OR SEVERE
 * PHYSICAL OR ENVIRONMENTAL DAMAGE ("HIGH RISK ACTIVITIES"). SUN
 * SPECIFICALLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR
 * HIGH RISK ACTIVITIES.
 */

package com.sun.commerce.gence;
import javax.commerce.util.AddressRecord;
import java.awt.*;
import java.net.URL;
import javax.commerce.util.Money;
import java.io.*;
import javax.commerce.base.*;
import javax.commerce.gui.*;
import javax.commerce.gui.image.*;
/**
 * A generic credit card object in the user's wallet.
 *
 * @author Daniel J. Guinan
 * @author Java Commerce Team
 * @version @(#)CCInstrument.java 1.21 11/07/97
 */
public class CCInstrument implements GenericCreditCard
{
  // The data that is to be stored for this instrument
  /** The Primary Account Number of this credit-card */
  protected String PAN = null;

  /** The expiration date of this credit-card */
  protected String expireDate = null;
  /** The description of this credit-card */
  protected String description = null;
  /** The name of the cardholder of this credit-card */
  protected String cardholderName = null;
  /** The billing address for this credit-card */
  protected AddressRecord billingAddress = null;
  /** The local alias (the name in the wallet) of this credit-card */
  protected String localAlias = null;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    /** The name of the image (taken from the /graphics directory in
     * the cassette */
    protected String imageName = null;
    // Declare any variables that are NOT to be stored with the
    // instrument as transient
    /** We keep the image around so that we don't have to re-create it */
    private transient Image theImage = null;
    /** This dummy frame is present to do image processing. AWT 1.1 requirement
   */
    private static transient Frame dummy = null;
    /** Constructor
     * @param desc The description of the card to find in the wallet
     * @param cardno The primary account number (the credit-card number)
     * @param expire The expiration date of the credit-card
     * @param holder The cardholder name
     * @param addr The card's billing address
     * @param gra The card's local alias
     */
    public CCInstrument (String desc,
            String cardno,
            String expire,
            String holder,
            AddressRecord addr,
            String gra,
            String imagenm)
    {
    description = desc;
    PAN = cardno;
    expireDate = expire;
    cardholderName = holder;
    billingAddress = addr;
    localAlias = gra;
    imageName = imagenm;
    }

    /**********************************************************************
     * Instrument interface specific methods
     **********************************************************************/
    /** Returns the description of the instrument
     * @return description
     */
    public String getDescription ()
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    { return description; }
    /** Sets the description of the instrument.
     * @param x The description of the credit-card.
     */
    public void setDescription (String x)
    { description = x; }
    /** Returns the context that this credit-card is generally used within.
     * @return always returns "pay" for generic credit cards
     */
    public String getContext() { return "pay"; }
  /** Sets the instrument type
     * @return In the case of Generic Credit Card, it always returns the
   * constant <tt>CCAdmin.Type_Name</tt>
     */
    public String  getType ()
    { return CCAdmin.TYPE_NAME; }
    /**
     * Returns the name the user associated with the specific instance
     * of the credit card
     * @return The local (wallet name) name of the credit-card.
     */
    public String getName()
    { return localAlias;}
    /**
     * Set the user-defined name of this instance of the credit card.
     * @param x The local alias (wallet name) name of the credit-card.
     */
    public void setName(String x)
    { localAlias = x; }
  /**
     * This method retrieves the visual representation of this instrument.
     * Since this method returns a Component, it can be active imagery
     * (e.g. an animation), or other such thing.
     *
     * @param media The CommerceContext used to fetch imagery from
     *    the jecf, a file, a URL, or the cassette.
     *
     * @param dim A hint as to the size of the image required.  Since
     *    proportions are important, we will not be returning
     *    an image exactly this size.  Rather, we will ensure
     *              that our returned image is proportionally correct,
     *              yet fits within these dimensions.
     *
     * @return Component A component with branding imagery associated with-
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
     *                    this instrument.
     */
    public Component getVisualRepresentation(CommerceContext media, Dimension dim)

    {
      try  // We will do the simplest thing - wrap the simple imagery
      {    // into a canvas and return it.
        Image img = getSimpleGraphic(media, dim);
        CWImage cimg = new CWImage(img,dim);
        //cimg.waitForDimensions();
        return cimg;

      } catch(Exception e) { return new Label("No Imagery"); }
    }
  /**
     * This method retrieves the image that is associated with this
     * instrument.
     *
     * @param media The CommerceContext used to fetch imagery from
     *                the jecf, a file, a URL, or the cassette.
     *
     * @param dim A hint as to the size of the image required.  Since
     *            proportions are important, we will not be returning
     *            an image exactly this size.  Rather, we will ensure
     *            that our returned image is proportionally correct,
     *            yet fits within these dimensions.
     *
     * @return Image The image representing this instrument.
     */
    public Image getSimpleGraphic(CommerceContext media, Dimension dim)
    {
      try
      {
        Image img,timg;
        if(dummy == null)          // We need a dummy frame to do image processing
        {                          // create it if we haven't already
          dummy = new Frame();
          dummy.addNotify();       // We need the Frame's peer to exist for
        }                          // this to work.. This forces that to happen.
        // We need a MediaTracker to ensure our processing results in
        // images that are ready for display
        //MediaTracker mt = new MediaTracker(dummy);
        if(theImage == null)  // If we haven't created the image, create it
        {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            timg = media.getImage(this,"graphics/" + imageName);
            //mt.addImage(timg,1);
            //mt.waitForID(1);

            // We will use the dummy frame to create a duplicate of the
            // image that we can do image processing on.
img = dummy.createImage(160,100);
Graphics g = img.getGraphics();
Color chromaKey = new Color(255,0,255);
g.setColor( chromaKey );
g.fillRect(0,0,160,100);
stampGraphic(g,160,100);
g.dispose();
theImage = ImageTools.mergeImages(timg,img,chromaKey);
        }

        return theImage;
    }
    catch(Exception e)
        { System.out.println(e); e.printStackTrace(System.out); return null; }
  }
  /**
   * This method writes out specific card related data on top of the
   * branding image, so the user can see the difference between two
   * instances of the same type of Credit Card.
   *
   * @param img The image to stamp with specific information
   */
  void stampGraphic(Graphics g, int w, int h)
  {
    try
    {
      Font f = new Font("Helvetica",Font.BOLD,11);
      g.setColor(Color.black);
      g.setFont(f);
      FontMetrics fm = g.getFontMetrics();
      Rectangle bounds = new Rectangle(0,0,w-1,h-1);
      //String drawStr = localAlias+"\n\n\n"+PAN+"\n"+expireDate+
      //"\n"+cardholderName;
      String drawStr = "\n\n\n"+PAN+"\n"-expireDate+"\n"+cardholderName;
      TextDraw.drawCentered(g,fm,drawStr,bounds,2,0,TextDraw.CENTERED);
      bounds = new Rectangle(1,1,w-1,h-1);
      g.setColor( new Color(254,254,254) );
      TextDraw.drawCentered(g,fm,drawStr,bounds,2,0,TextDraw.CENTERED);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
        catch(Exception e)
          { System.out.println(e); e.printStackTrace(System.out); }
      }
      /*****************************************************
      **************
       * GenericCreditCard interface specific methods
       ****************************************************
      *************/
      /* Check if this instrument allows this purchase
       * Has the option of returning false and keeping all data un - available
       * In this type of seneraio, the data would be unavailable by default,
       * and would only become available if a valid accept() occurs...
       * (e.g. * having a boolean OKAYTOGIVEOUTINFO; variable that is checked
       * by each getter().. set OKAYTOGIVEOUTINFO = true; only if accept
       * succeeds).
       *
       * @return true = is acceptable, false = is not acceptable
       */
      public boolean accept()
      {
        // Has the option of returning false and keeping all data unavailable
        // In this type of scenario, the data would be unavailable by default,
        // and would only become available if a valid accept() occurs...
        // (e.g. -- having a boolean OKAYTOGIVEOUTINFO; variable that is checked
        // by each getter().. set OKAYTOGIVEOUTINFO = true; only if accept
        // succeeds).
        return true;
      }
      /**
       * Returns the credit - card number
       * @return the Instrument's card number
       */
      public String getPAN()
      { return PAN; }
      /**
       * Returns the card's expiration date
       * @return the Instrument's expire date
       */
      public String getExpireDate()
      { return expireDate; }
      /**
       * Returns the cardholder's name as known by the issuing institution
       * @return the Instrument's cardholder name
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间：可以联系群主获取更多大型企业内部技术教程。

```
     */
    public String getCardholderName()
    { return cardholderName; }
    /**
     * Return's the cardholder's billing address
     * @return the Instrument's cardholder address
     */
    public AddressRecord getBillingAddress()
    { return billingAddress; }

    /**
     * Sets the credit - card number for this instance
     * @param the card number in string format
     */
    public void setPAN(String x)
    { PAN = x; }
    /**
     * sets the card's expiration date
     * @param the expiration date as a string
     */
    public void setExpireDate(String x)
    { expireDate = x; }
    /** * Sets the cardholder's name as known by the issuing institution
     * the cardholder's name
     */
    public void setCardholderName (String x)
    { cardholderName = x; }
    /**
     *   Sets the billing address as an AddressRecord
     * @param The billing address as an AddressRecord
     */
    public void setBillingAddress (AddressRecord x)
    { billingAddress = x; }
} // end of CCInstrument
```

45.3.3 协议盒式带

协议盒式带处理客户与服务器之间的通信。协议盒式带的一个例子是处理 SET(Secure Electronic Transaction)协议的协议盒式带。由于协议的变化很大,所以不同的协议盒式带之间基本没有什么共同点。Protocol 接口定义了一些所有协议盒式带必须共享的方法。

如果一个协议支持某种特殊的工具,则 canUseInstrument 方法将返回 true(真):

```
public boolean canUseInstrument(Instrument instrument)
```

在确定是否支持一个工具时,Sun 公司建议使用下面的代码结构:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
if(! (instrument instanceof NeededInterface_1))
    return false;

if(! (instrument instanceof NeededInterface_2))
    return false;

if(! (instrument instanceof NeededInterface_3))
    return false;

return true;
```

当使用控制盒式带注册之后，getName 方法将返回协议的名字：

```
public String getName( )
```

setProtocolJCM 通知协议盒式带读一个 JCM 并提取信息：

```
public void setProtocolJCM(JCM protocolJCM)
```

setWalletGate 方法给协议一个 WalletGate 对象，它用来获取执行某种特殊操作的许可：

```
public void setWalletGate(WalletGate gate)
```

据 Sun 公司称，这种操作许可和 JCM 的协议部分可能会很快合并到 CommerceContext 对象中，所以在 JCC API 的未来版本中 setProtocolJCM 和 setWalletGate 方法可能不会再出现。

setCommerceContext 方法为当前协议指定一个商业环境：

```
public void setCommerceContext(CommerContext context)
```

商业环境中包含有针对当前操作的信息。Sun 公司预计在 JCC 将来的版本中，在创建协议时就进行商业环境的指定，取消 setCommerceContext 方法。

PurchaseProtocol 接口定义了一个典型的用于进行买入的协议。在电子环境下与实际的购买过程一样，PurchaseProtocol 接口只包含一个方法：

```
public boolean actUpon(Instrument instrument,
    PurchaseParams purchase) throws TransactionException
```

程序清单 45.2 给出了一个协议实例，它来自于 Sun 公司提供的 JCC 包。

```
/*
 * @(#) @(#)DemoProtocol.java 1.7 11/07/97
 *
 * Copyright (c) 1996 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Permission to use, copy, modify, and distribute this software
 * and its documentation for NON-COMMERCIAL or COMMERCIAL purposes and
 * without fee is hereby granted.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
 * Please refer to the file http://java.sun.com/copy_trademarks.html
 * for further important copyright and trademark information and to
 * http://java.sun.com/licensing.html for further important licensing
 * information for the Java (tm) Technology.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
 * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
 * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR
 * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
 * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
 *
 * THIS SOFTWARE IS NOT DESIGNED OR INTENDED FOR USE OR RESALE AS ON-LINE
 * CONTROL EQUIPMENT IN HAZARDOUS ENVIRONMENTS REQUIRING FAIL-SAFE
 * PERFORMANCE, SUCH AS IN THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT
 * NAVIGATION OR COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL, DIRECT LIFE
 * SUPPORT MACHINES, OR WEAPONS SYSTEMS, IN WHICH THE FAILURE OF THE
 * SOFTWARE COULD LEAD DIRECTLY TO DEATH, PERSONAL INJURY, OR SEVERE
 * PHYSICAL OR ENVIRONMENTAL DAMAGE ("HIGH RISK ACTIVITIES"). SUN
 * SPECIFICALLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR
 * HIGH RISK ACTIVITIES.
 */
// Things to be done to this code are marked with ?? or UNDONE:
package com.sun.commerce.example.demoprot;

import java.awt.*;
import java.awt.Event;
import java.io.*;
import java.net.*;
import java.security.*;
import java.util.*;

import javax.commerce.base.*;
import javax.commerce.base.WalletUserPermit;
import javax.commerce.cassette.*;
import javax.commerce.database.*;
import com.sun.commerce.gencc.GenericCreditCard;
import javax.commerce.gui.ProgressBar;
import javax.commerce.util.Money;

/**
 * The DemoProtocol is a protocol that is intended to be used for
 * Demo purposes.  It does not actually perform a transaction or
 * open any network connections.  It mostly puts up a progress
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
 * bar and pretends that it performed a transaction.  This protocol
 * will accept any instrument.
 *
 * @see PurchaseProtocol
 * @see ActionBase
 * @see ActionParameter
 */
public class DemoProtocol implements PurchaseProtocol, Runnable
{
/** The registration name of the protocol */
public static final String PROTOCOL_NAME = "Demo";

/** The instrument being used in the demo protocol */
Instrument instrument;

private PurchaseParams pp;

private ProgressBar pbar;

private JCM jcm = null;

private String failure = null;

private boolean protocolDone = false;

private CommerceContext media;

/** Constructor
  */
public DemoProtocol() {}

/**
  * Returns the registration name of the Demo Protocol.
  *
  * @return String The registration name of the Demo Protocol.
  */
public String getName() { return PROTOCOL_NAME; }

/**
  * Called by the Operation to set the protocol's JCM.
  *
  * @param prjcm The part of the JCM that applies to the protocol.
  */
public void setProtocolJCM(JCM prjcm) { jcm = prjcm; }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public void setWalletGate(WalletGate wg)
{}

  public void setCommerceContext(CommerceContext ccntxt)
  {
    media = ccntxt;
  }
  /**
   * Pretends to perform the transaction.  Uses the passed instrument and
   * parameters present a convincing progress bar.
   *
   * @param inst The CreditCard instrument used for the transaction
   *
   * @param ap The parameters used for the payment
   *
   * @return boolean true = success
   */
synchronized public boolean actUpon(Instrument inst, PurchaseParams pp)
  throws TransactionException
{
  // record parameters and double-check the instrument is of
  // the correct type.

  this.pp = pp;

  pbar = new ProgressBar(media,"Paying" +
    ((javax.commerce.util.Invoice)pp.getInvoice()).getTotal().toString() +
    "using " + pp.getInstrument().instrument.getDescription());
  pbar.setTitle(pp.getMerchant().getGeneralURL().toString());

  instrument = inst;
  if (true) {
   System.out.println("Running Protocol in the same thread.");
   commit();
  } else {
   System.out.println("Running Protocol as a sepparate thread.");
   // Start the thread that will monitor the transaction
   Thread t = JECF.makeThread(this);
   t.start();
   while (! protocolDone) {
    try { wait();
    } catch (InterruptedException ignored) {}
   }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }

    if (failure != null) {
      System.out.println("Failure:" + failure);
      throw new TransactionException(failure);
    }

    // Return DONE -> An exception will be raised for an error condition
    return true;
  }

  /**
   * Check to see if this Protocol can use a specific instrument
   *
   * @param instrument The instrument to check
   *
   * @return boolean true = can use, false = cannot use
   */
  public boolean canUseInstrument(Instrument instrument)
  {
    return true;
  }

  private void pause(int milliseconds) {
    Thread thisThread = Thread.currentThread();
    try { thisThread.sleep(500);
    } catch (InterruptedException ignored) {}
  }

  private void setFailure(String newFailure) {
    failure = newFailure;
    if (newFailure != null) {
      System.out.println("setFailure(" + failure + ")");
      throw new RuntimeException(failure);
    }
  }

  private void checkCancelled (boolean cancelled) {
    if (cancelled) {
      setFailure("User Cancelled Transaction");
    }
  }

  private void getJcmInfo() {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    if (jcm! = null)
     try { setFailure(jcm.getString("failure")); }
     catch (JCMException ignored) { }
    if (jcm! = null) {
     System.out.println("Displaying JCM:");
     Enumeration leaves = jcm.elements();
     while (leaves.hasMoreElements()) {
      String[] leaf = (String[]) leaves.nextElement();
      if (leaf ! = null) {
       System.out.print("    " + leaf[0] + "=");
       for (int i = 1; i < leaf.length; i++) {
        System.out.print(leaf[i] + "  ");
       }
       System.out.println("");
      }
     }
    } else {
     System.out.println("No JCM was set!");
    }
  }

  private Random rand = new Random();
  int rand(int minValue, int maxValue) {
    return minValue + Math.abs(rand.nextInt()%(maxValue - minValue));
  }

  synchronized protected boolean commit()
  {
    try {
     getJcmInfo();
     // Start the progress bar
     pause(500);
     checkCancelled(pbar.showProgress (0, "Initiating Communications"));
     for (int i = rand(5,15); i < 100; i += rand(5,15)) {
      pause(rand(10,500));
      checkCancelled(pbar.showProgress (i, "Transaction is " + i + "% done."));
     }
     pause(500);
     pbar.showProgress (100,"Transaction is complete.");

     return true;
    } catch (RuntimeException ignored) {
     return false;
    } finally {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        protocolDone = true;
        notify();
        pbar.done ();
      }
    }

    public void run(){
      commit();
    }
  }
```

45.3.4　操作盒式带

操作盒式带实际上是 JCC 的核心。当选择一个操作后，该操作的用户接口收集完成这一交易所需要的信息，获取必要的工具并激活一个协议来完成这一交易。

> **注意**：与 JavaCommerceClient 的其余部分一样，本节中的信息也是要改变的。Sun 公司提出操作接口中的某些信息在将来的版本中将合并到 CommerceContext 对象中。

操作接口及其有关的接口表示了一个非常难以适应的复杂系统。操作接口中的大部分方法用于为操作设置初始的上下文关系。setJCM 方法传送给操作对象一个 JCM，以分析和检索与该操作有关的信息：

```
public void setJCM (JCM message) throws Exception
```

getJCMDescription 方法为该操作返回必选和可选参数的描述：

```
public String[ ] getJCMDescription( )
```

描述字符串以“name = description(名字 = 描述)”的形式返回。

SetWalletGate 方法传送给该操作一个 WalletGate，用于获得各种安全许可：

```
public void setWalletGate(WalletGate gate)
```

SetID 方法传送给该操作它在数据库中的行 ID：

```
public void setID (RowID id)
```

setCommerceContext 方法设置该操作的当前上下文关系：

```
public void setCommerceContext(CommerceContext context)
```

将来，有些其它参数或许会合并到商业上下文关系中，上下文关系本身也可能成为该操作的构造器的一部分。

execute 方法用来启动操作，该操作启动后会产生一个 UI 对象，收集有关该操作的信息：

```
public String execute( ) throws Exception
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

45.3.5 服务盒式带

服务盒式带是 JCC 中其它盒式带所使用的一个工具。不存在服务接口功能，但是有一个 ServiceUI，因为服务盒式带要执行用户接口功能。另外，ServiceUI 接口由操作接口使用于用户接口。

ServiceUI 接口中的 getClientContainer 方法返回表示用户接口的容器：

```
public Container getClientContainer(CommerceContext  context, Dimension dim)
```

getSelectedImage 和 getUnselectedImage 方法返回表示该服务的图像，这些图像作为可选择对象放在另一个容器内：

```
public Image getSelectedImage(CommerceContext context, Dimension dim)
public Image getUnselectedImage(CommerceContext context, Dimension dim)
```

getSelectorText 返回一个图像显示时作为其标签的文本：

```
public String getSelectorText()
```

最后，setWalletGate 方法传送给 ServiceUI 一个 WalletGate 对象，用于安全操作：

```
public void setWalletGate(WalletGate gate)
```

程序清单 45.3 给出了一个 Rolodex 服务实例，该服务实例来自于 Sun 公司的 JCC 包。

```
/*
 * @(#)Rolodex.java      1.1 97/10/29
 *
 * Copyright (c) 1996 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Permission to use, copy, modify, and distribute this software
 * and its documentation for NON-COMMERCIAL or COMMERCIAL purposes and
 * without fee is hereby granted.
 * Please refer to the file http://java.sun.com/copy_trademarks.html
 * for further important copyright and trademark information and to
 * http://java.sun.com/licensing.html for further important licensing
 * information for the Java (tm) Technology.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
 * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
 * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR
 * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
 * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
 *
 * THIS SOFTWARE IS NOT DESIGNED OR INTENDED FOR USE OR RESALE AS ON-LINE
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
 * CONTROL EQUIPMENT IN HAZARDOUS ENVIRONMENTS REQUIRING FAIL-SAFE
 * PERFORMANCE, SUCH AS IN THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT
 * NAVIGATION OR COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL, DIRECT LIFE
 * SUPPORT MACHINES, OR WEAPONS SYSTEMS, IN WHICH THE FAILURE OF THE
 * SOFTWARE COULD LEAD DIRECTLY TO DEATH, PERSONAL INJURY, OR SEVERE
 * PHYSICAL OR ENVIRONMENTAL DAMAGE ("HIGH RISK ACTIVITIES"). SUN
 * SPECIFICALLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR
 * HIGH RISK ACTIVITIES.
 */
package com.sun.commerce.example.rolodex ;
import java.awt.*;
import javax.commerce.database.* ;
import javax.commerce.cassette.* ;
import javax.commerce.util.*;
import javax.commerce.base.*;
import java.util.* ;
import java.io.* ;
import java.security.* ;
import javax.commerce.base.Constants;
import javax.commerce.gui.InfoDialog;

/**
 * @author Surya Koneru
 * @(#)Rolodex.java      1.1 97/10/29
 */
public class Rolodex implements ServiceUI
{
  public static final String SERVICE_NAME = new String("Rolodex");


  public WalletGate wgate;
  private WalletAdminPermit wap;
  private DatabaseOwnerPermit dop;

  /**
   * This method will always be called before any other ServiceUI methods
   *
   * @param gate A WalletGate to allow the service to utilize wallet-level
   *             functionality.
   */
  public void setWalletGate(WalletGate gate)
  {
    wgate = gate;
    wap = gate.getWalletAdminPermit( new Ticket("W_OWNER"));
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    dop = wap.getDatabaseOwnerPermit();
  }
  /**
   * This method fetches a service's client container for display within
   * the encompassing WalletUI.  The ServiceUI author is urged to use
   * commerce widgets, but it is not neccessary. <br> <br>
   *
   * This method, in general, returns an a light - weight container that
   * has transparent characteristics.  It may,
   * however, return any Container.
   *
   * @param ccontext A CommerceContext that may be used to fetch imagery
   *                 and for use in constructing the ServiceUI. The
   *                 UIFactory of this CommerceContext models the
   *                 current WalletUI.
   *
   * @param hint The initial dimensions of the Service UI.  The returned
   *             container is likely to be resized many times, however.
   *
   * @return Container The AWT Container that represents the Service's UI.
   *                   This Container should have dimensions that match
   *                   the passed in hint.
   */
  public Container getClientContainer(CommerceContext ccontext, Dimension hint)
  {
    return new RolodexPanel(ccontext,dop);
  }

  /**
   * Used to fetch the image displayed on the WalletUI selector when the
   * selector for this service is not selected.
   *
   * @param factory A CommerceContext that may be used to fetch imagery.
   * @param size The size of the requested image (Generally 16x16 pixels)
   *
   * @return Image The image to display when this serivce is not selected.
   */
  public Image getUnselectedImage(CommerceContext factory, Dimension size)
  { return null; }

  /**
   * Used to fetch the image displayed on the WalletUI selector when the
   * selector for this service is selected.
   *
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
     * @param factory A CommerceContext that may be used to fetch imagery.
     * @param size The size of the requested image (Generally 16x16 pixels)
     *
     * @return Image * The image to display when this service is selected.
     */
    public Image getSelectedImage(CommerceContext factory, Dimension size)
    { return null; }

    /**
     * Used to fetch the text that will be displayed on the WalletUI selector.
     *
     * @return String -- The text to display on the WalletUI selector for this
     *                   service.
     */
    public String getSelectorText() { return SERVICE_NAME; }
}
```

45.3.6　用户接口盒式带

用户接口盒式带包括一个 WalletUI 对象，一个 TransactionListener 、一个 ActionListener 和一个 UIFactory。

WalletUI 接口定义了一组在该 UI 中可使用的操作和服务，以及有关当前安全性上下文关系的方法。addOperation 方法向这一用户接口可用的操作中添加一个操作，并返回一个唯一索引值：

```
public int addOperation(Operation op)
```

当用户接口盒式带支持一个特殊操作时 CanUseOperation 方法返回 true(真)：

```
public boolean canUseOperation(Operation op)
```

addSelector 方法将一个服务增加到可用服务中，并返回一个唯一索引值：

```
public int addSelector(ServiceUI service)
```

removeSelector 方法从用户接口中删去一个服务：

```
public void removeSelector(int index)
```

addSelector 和 removeSelector 方法处理用户接口中的可视对象。如果添加了一个具有用户接口组件的操作，addOperation 方法有可能调用 addSelector 方法将其增加到可视项目中。如果一个操作有可视组件，可使用 removeSelector 方法将其删除，因为此处没有 removeOperation 方法。

SetCommerceContext 方法为用户接口设置当前上下文关系，并会在用户接口实例化之后，立即得到调用：

```
public void SetCommerceContext(CommerceContext context)
```

init 方法必须在商业上下文关系设置之后，其它任何操作执行之前调用：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public void init( )
```

getName 方法返回与 CassetteControl 类登记时相同的用户接口盒式带名：

```
public String getName( )
```

select 方法在用户接口中选择一个不同的对象：

```
public void select(int selectorIndex)
```

选择值应该是可用选择对象之一的唯一索引值。

populate 方法通知用户接口在一个 AWT 容器中绘制自己。

```
public void populate(Container cont)
```

切记，populate 方法有可能在一个可视容器上进行操作。所以，如果我们要进行一些不美观的操作，如增加一组对象，然后又删除这些对象，用户界面看上去就会有些古怪。

程序清单 45.4 给出了一个用户接口盒式带演示，它来自于 Sun 公司的 JCC 实例。

```
/*
 * @( # )DemoUI.java 1.23 11/07/97
 *
 * Copyright (c) 1997 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Permission to use, copy, modify, and distribute this software
 * and its documentation for NON - COMMERCIAL or COMMERCIAL purposes and
 * without fee is hereby granted.
 * Please refer to the file http://java.sun.com/copy _ trademarks.html
 * for further important copyright and trademark information and to
 * http://java.sun.com/licensing.html for further important licensing
 * information for the Java (tm) Technology.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
 * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
 * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, OR NON - INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR
 * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
 * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
 *
 * THIS SOFTWARE IS NOT DESIGNED OR INTENDED FOR USE OR RESALE AS ON - LINE
 * CONTROL EQUIPMENT IN HAZARDOUS ENVIRONMENTS REQUIRING FAIL - SAFE
 * PERFORMANCE, SUCH AS IN THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT
 * NAVIGATION OR COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL, DIRECT LIFE
 * SUPPORT MACHINES, OR WEAPONS SYSTEMS, IN WHICH THE FAILURE OF THE
 * SOFTWARE COULD LEAD DIRECTLY TO DEATH, PERSONAL INJURY, OR SEVERE
 * PHYSICAL OR ENVIRONMENTAL DAMAGE ("HIGH RISK ACTIVITIES"). SUN
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
 * SPECIFICALLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR
 * HIGH RISK ACTIVITIES.
 */
package com.sun.commerce.example.demoui ;
import javax.commerce.database.* ;
import javax.commerce.cassette.* ;
import javax.commerce.util.* ;
import java.util.* ;
import java.io.* ;
import java.security.* ;
import javax.commerce.base.* ;
import javax.commerce.gui.* ;
import java.awt.* ;
import java.awt.event.* ;

/**
 * @author Daniel J. Guinan
 * @version @(#)DemoUI.java 1.23 11/07/97
 *
 */
public class DemoUI implements WalletUI, ActionListener, TransactionListener
{
  class ServiceNode
  {
    public ServiceUI service;
    public Container container;
    public CWSelector button;
    public int selectorIndex;

    public ServiceNode(ServiceUI sui, CommerceContext media,
        String buttontext, int index)
    {
      service = sui;
      container = null;
      selectorIndex = index;
      Dimension d = new Dimension(20,20);
      button = new CWSelector(media,buttontext,
       sui.getUnselectedImage(media,d),
       sui.getSelectedImage(media,d));
    }
  }

  public static final String WALLETUI_NAME = "DemoUI";
  public static final String WALLETUI_DESCRIPT = "Demonstration Wallet User Interface";
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public static final Color INACTIVE_COLOR = Color.lightGray;
    public static final Color ACTIVE_COLOR = new Color(150,175,255);

    private CommerceContext context;
    private UIFactory widgets;

    private CWPanel loadingPanel;
    private CWPanel selectionPanel;
    private Container currentClient = null;
    private CWPanel masterContainer = null;
    //  private CWSelector selectedButton = null;
    private ServiceNode currentlySelected = null;

    private Hashtable serviceTable = new Hashtable();  // (name, ServiceNode)
    private Hashtable idxMapping = new Hashtable();    // (Integer, name)
    private Hashtable pendingTable = new Hashtable();  // (Operation, name)

    private int currentIdx = 0;

    ////////////////////////////////////////////////////////////////
    ////////////////// WalletUI Interface Methods //////////////////
    ////////////////////////////////////////////////////////////////

  /**
     * Init() is called after the current CommerceContext and
     * CommerceUIFactory objects are set, but before any other
     * calls are made into the WalletUI.  It is here that initialization
     * code should be executed. <br> <br>
     *
     * PLEASE NOTE:  This is not where the UI is shown.  That occurs
     * in the populate() method.
     */
    public void init()
    {

    }

    /**
     * Retrieves the current CommerceContext.
     *
     * @return CommerceContext -- The current CommerceContext
     */
    public CommerceContext getCommerceContext()
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    { return context; }

    /**
     * Sets the current CommerceContext (THIS METHOD IS ALWAYS
     * CALLED DIRECTLY AFTER OBJECT INSTANTIATION)
     *
     * @param ccontext The CommerceContext to use as current.
     */
    public void setCommerceContext(CommerceContext ccontext)
    {
      context = ccontext;
      new CommonGraphics(ccontext);  // Load static common graphics...
      widgets = new DemoUIFactory();
      context.setUIFactory( widgets );

      loadingPanel = new CWPanel(ccontext);
      loadingPanel.setLayout(new BorderLayout());
      currentClient = loadingPanel;
      CWLabel loading = new CWLabel(ccontext,"Loading...",CWLabel.CENTER);
      loading.setFont(new Font("GilSans",Font.BOLD,20));
      loading.setForeground(Color.white);
      loadingPanel.add(loading,BorderLayout.CENTER);
    }

    /**
     * This method is called to add a Service selector.
     *
     * @param service The ServiceUI being added.
     * @return int -- The unique index of the selector created.
     */
    public int addSelector(ServiceUI service)
    {
System.out.println("DEMOUI: Adding service = "+service.getSelectorText());

      int ret = currentIdx++;
      String selectorText = service.getSelectorText();

      //UNDONE: Freak out if the selectorText is already there..
      if( serviceTable.get(selectorText) != null )
        {
boolean unique = false;
int index = 1;
String name = null;
while(! unique)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }
      name = selectorText + " " + ( + + index);
      if(serviceTable.get(name) = = null) unique = true;
    }
selectorText = name;
      }
    //return -1;

    idxMapping.put( new Integer(ret), selectorText );

    ServiceNode sn = new ServiceNode(service,context,selectorText,ret);
    serviceTable.put(selectorText, sn);
    selectionPanel.add(sn.button);
    sn.button.addActionListener(this);
    // Do we really want to do this??
    selectionPanel.validateAll();

    return ret;
  }

  /* *
   *
   * @param op The Operation that should have a selector added for it.
   * @return int * The unique index of the selector created.
   */
  public int addOperation(Operation op)
  {
    int idx = -1;
    if(op instanceof ServiceUI)
      {
idx = addSelector((ServiceUI)op);
select(idx);
String name = (String)idxMapping.get( new Integer(idx) );
ServiceNode sn = (ServiceNode)serviceTable.get(name);
sn.button.setPending(true);
pendingTable.put(op,name);
op.addTransactionListener(this);
      }

    //UNDONE: Exception??
    return idx;
  }

  public void transactionPerformed(TransactionEvent evt)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    {
      Operation source = (Operation)evt.getSource();
      String name = (String)pendingTable.get(source);
      if(name = = null) return;
      pendingTable.remove(source);
      ServiceNode sn = (ServiceNode)serviceTable.get(name);
      sn.button.setPending(false);
    }

    /* *
     * This method is called to remove a Service selector.
     *
     * @param int The unique index of the Service selector to remove.  This
     *            is the same index returned by addSelector().
     */
    public void removeSelector(int idx)
    {
      Integer i = new Integer(idx);
      String stext = (String)idxMapping.get(i);
      if(stext = = null)
        {
System.out.println("Cannot remove selector: Index " + idx +
        "is not valid!");
return; //Not there??
        }

      ServiceNode sn = (ServiceNode)serviceTable.get(stext);
System.out.println(" > > Removing Service " + stext);
//      selectionPanel.setCanInvalidate(false);
      selectionPanel.remove(sn.button);
      sn.button.removeActionListener(this);
      idxMapping.remove(i);
      serviceTable.remove(stext);
      selectionPanel.validateAll(); // we removed a button...
      if(sn.selectorIndex = = idx)
      {
        //If this guy is currently selected, change...
        Enumeration e = serviceTable.elements();
        sn = (ServiceNode)e.nextElement();
        if(sn! = null) select(sn.selectorIndex);
        else showClient(null);
      }
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
/**
 * This method is called to force the WalletUI to change it's focus to
 * a particular service.
 *
 * @param idx The unique index of the Service selector to select.  This
 *            is the same index returned by addSelector().
 */
public void select(int idx)
{
  Integer i = new Integer(idx);
  String stext = (String)idxMapping.get(i);
  if(stext == null) return; // Not there??

  showClient(stext);
}
/**
 * This method is used to determine compatibility with Operations.  When
 * Operations or WalletUIs are installed, this method is called on all
 * WalletUIs against all Operations to generate a compatibility list.
 * <br><br>
 *
 * It is this compatibility list that appears in the user's preferences,
 * allowing a user to change their "preferred" UI for various operations.
 * <br><br>
 *
 * In general, this method will check interfaces of the Operation and return
 * true if it feels that it can accomidate the Operation's UI requirements.
 * <br><br>
 *
 * @param op The operation to check for UI compatibility.
 * @return boolean * true=compatible with operation, false=not compatible
 *
 * @see WalletUI.addOperation
 */
public boolean canUseOperation(Operation op)
{
  //The demonstration UI works with any operation that implements ServiceUI...
  if(op instanceof ServiceUI) return true;
  else return false;
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
/**
 * Retrieves the registration name of this WalletUI.  This name must match
 * the registration name used in the CassetteControl.install() method when
 * registering this UI.
 *
 * @return String -- The registration name of this WalletUI.
 */
public String getName()
{
  return WALLETUI_NAME;
}



/**
 * This method is called to present the WalletUI.  In general, this happens
 * right before the JECF performs a show() on the frame that contains the
 * various widgets that represent the WalletUI.  The whole of the WalletUI
 * will be shown within the container passed in this method.  The following
 * considerations should be carefully taken into account by WalletUI cassette
 * writers: <br> <br>
 *
 * <li> The container passed to this method is probably, but not
 *       necessarily a Frame or an Object that inherets Frame.
 *       (NOTE: Never cast this Container to Frame)
 *
 * <li> The container may change dimensions.  It is expected that the
 *       User will resize the Window, causing this container to resize
 *       as a result.  The UI must be written to accommodate this (i.e.
 *       stretch).
 *
 * <li> Do not rely on any characteristics of any AWT components that
 *       may exist above this Container.  Those characteristics are
 *       subject to change at any time, forcing any cassettes that rely
 *       upon them to become obsolete.
 *
 * <li> The populate method may be populating a live AWT component
 *       (i.e. already shown).
 * </ul>
 *
 * @param c The container that the WalletUI should be drawn within.
 */
public void populate(Container c)
{
  masterContainer = new CWPanel(context);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    c.setLayout(new BorderLayout(0,0));
    c.add(masterContainer,BorderLayout.CENTER);
    masterContainer.setLayout(new BorderLayout(0,0));

    currentClient = loadingPanel;
    masterContainer.add(currentClient,BorderLayout.CENTER);
    loadingPanel.validateAll();

    selectionPanel = new SelectionPanel(context);
    selectionPanel.setLayout( new VFlowLayout(VFlowLayout.TOP|VFlowLayout.HORZ_LEFT) );

    masterContainer.add(selectionPanel,BorderLayout.EAST);
}

//////////////////////////////////////////////////////////////////
////////////////////// Private Methods //////////////////////////
//////////////////////////////////////////////////////////////////

public void actionPerformed(ActionEvent evt)
{
    Object source = evt.getSource();
    if(source instanceof CWSelector)
    {
        CWSelector bt = (CWSelector)source;

        String item = bt.getLabel();
        showClient(item);
        //cardLayout.show(switchPanel,item);
    }
}

private void showClient(String stext)
{
    Container newClient;
    // Tell the selector it is not current anymore.
    if(currentlySelected! = null) currentlySelected.button.setCurrent(false);

    if(stext = = null) // What is this??
    {
        newClient = loadingPanel;
        currentlySelected = null;
    }
    else // Get the new client
    {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    ServiceNode sn = (ServiceNode)serviceTable.get(stext);
    newClient = sn.container;
    currentlySelected = sn;
    // Tell the new selector it is not current anymore.
    sn.button.setCurrent(true);
  }

  // If this is the first time, do something special.
  if(newClient == null) firstTimeShowClient(stext);
  else // Just change the page
  {
    //if(currentClient! = null)
    masterContainer.swapComponentValid(currentClient,newClient,
    BorderLayout.CENTER);
    currentClient = newClient;
  }

  //reLayoutPaintMaster();
}

/*   void reLayoutPaintMaster()
{
  // Relayout and repaint everything...
  if(masterContainer instanceof CWBasePanel)
    ((CWBasePanel)masterContainer).validateAll();
  else if(masterContainer instanceof CWidget)
    ((CWidget)masterContainer).validateAll();
  else { masterContainer.validate(); masterContainer.repaint(); }
}*/

private void firstTimeShowClient(String stext)
{
  // Show the loading loadingPanel
  if(currentClient! = loadingPanel)
  {
    masterContainer.swapComponentValid(currentClient,loadingPanel,
    BorderLayout.CENTER);
    currentClient = loadingPanel;
    context.showStatus("Loading...");
  }

  // Do the normal stuff...
  ServiceNode sn = (ServiceNode)serviceTable.get(stext);
  Container clientArea =
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        sn.service.getClientContainer(context,loadingPanel.getSize());
        clientArea.setSize(loadingPanel.getSize());
        clientArea.setLocation(loadingPanel.getLocation());
        sn.container = clientArea;

        // swap
        masterContainer.swapComponentValid(currentClient,clientArea,
                BorderLayout.CENTER);
        currentClient = clientArea;
        context.showStatus("");
    }

}
```

Java Commerce Client 是一个很新的 API,在其实际成为 Internet 上的一个有影响的产品之前,或许还要经过若干次的修改。虽然本章已经全面介绍了商业客户对象的结构,我们或许仍不必编写自己的商业对象。相反,我们应该能够从各软件商和在线存储中获取盒式带。然而,如果我们正在创建自己的购货卡小程序,则可能需要编写自己的 UI 盒式带,把底层盒式带的开发留给其它开发商。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 46 章　数据结构及 Java 工具

本章内容：

46.1　什么是数据结构
46.2　集成
46.3　Vector 类
46.4　Hashtable 类
46.5　Properties 类
46.6　Stack 类
46.7　Date 类
46.8　BitSet 类
46.9　StringTokenizer 类
46.10　Random 类
46.11　Observable 类

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

46.1 什么是数据结构

Java.util 包提供了若干个有用的类,这些类为 Java 运行环境增添了重要的功能。这些类为用户提供了在 C++下经常要由用户自己编写的大量代码。Java 的开发者认识到,人们真正喜欢 Smalltalk 的原因之一就是,它有丰富的实用工具类。

Java.util 包的重点在于容器对象——即包含或拥有其它对象的对象。除了容器之外,该软件包还增加了一个方便的工具类,包括分解字符串成为单词(标志),扩展对随机数及日期的支持,以及一个对 Smalltalk 的补充,称为可观察类。

数据结构是一般计算机科学中用于描述一个由一组其它对象组成的对象的术语。例如,数组就是一个最简单的数据结构。不同数据结构的复杂程度也不同,但其最终的目的都是拥有和操作对象。

有些数据结构如数组以一个长列表的形式组织数据,而另一些数据结构如树,则以非线性的存储单元顺序组织数据。每种类型的数据结构都有其优点和缺点。例如,树结构对于查找和插入有顺数据的操作非常有效,而散列表对于查找数据更为有效,但它要占用较多的内存空间。

46.2 集成

Java 2API 设计上的一个重要变化是创建了一个由一组类组成的称为集成 API(Collection API)的东西。集成 API 为 java.util 包中的所有数据结构提供了一组公共的接口。

在集成之前,从一种数据结构到另一种数据结构的转换需要做相当多的工作。而集成 API 为这些操作提供了统一的机制。另外,集成 API 的设计允许一些特性如对象的排序,使用于许多类型的数据结构。

这样我们就可以根据其执行特性选择适当的数据结构,而不必担心其实现机制了。

46.2.1 集成接口

集成 API 的根是一个集成(Collection)接口。集成接口的目的是提供所有集成类都具有的所有通用方法。目前,并非所有集成都全部提供了其方法的实现。如果试图调用集成中一个未实现的方法。

则该方法将可能产生一个 UnsuportedOperationException 异常。

正如我们所希望的,集成接口提供了向集成中插入新对象的机制。第一个方法允许用户插入一个单个的对象。另外一个方法允许用户添加另一个集成中的所有元素。

```
boolean add(Object o)
boolean addAll(Collection c)
```

为了从一个集成中取出一个对象,可选择以下三种方法。前两个方法使用一个数组返回容器的内容。有意思的是在第二个方法中我们传送的对象类型是一个数组,这也正是我们所希望的返回类型。最后一个方法返回一个新的类称为 Iterator。我们将在本章稍后讨论 Iterator 的使用方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Object[] toArray()
Object[] toArray(Object[ ] a)
Iterator iterator()
```

我们也可以从集成中删除对象。我们可以使用清空数组的方法(clear( )),也可以使用删除一个指定对象的方法(remove( ))。另外,我们可以通过传送一个集成参数,删除该集成中已经存在的所有对象的方法,删除一组对象;也可以通过传送一个集成参数,保留该集成中的对象,而删除其余的元素。

```
void clear()
boolean remove(Object o)
boolean removeAll(Collection c)
boolean retainAll(Collection c)
```

最后一组方法允许我们检查集成的状态。前两个方法允许我们确定一个集成中是否包含一个对象或一组对象。第三个方法用来确定集成是否为空,第四个方法则返回集成中元素的个数。

```
boolean contains(Object o)
boolean containsAll(Collection c)
boolean isEmpty()
int size()
```

集成接口是由本章所介绍的一些类实现的,稍后我们将通过实例说明如何使用这些方法。

46.2.2　List 接口

集成接口使用两个子接口 List(列表) 和 Map(映射)进行扩展和特例化。列表接口建立于集成之上,并向有序存放对象的集成添加了一些方法。

> **注意**:在 Java 1.2 API 中创建的 Java.util.List 接口,使用了一个与 java.awt.List 冲突的名字。如果同时导入 java.awt.* 和 java.util.*,我们必须对使用 List 做一些特殊处理。有两种处理方式:通过特别地导入 java.util.List,或通过使用 List 的全名(java.util.List)而不是只使用 List。

有序列表增加的主要概念是索引概念。索引是对象实际所存储的位置。所以,列表接口增加了几个用来处理索引概念的方法。

两个新的 add()方法允许用户在一个特定的索引位置插入元素,并移动该索引位置以后的元素,使集成中的其余对象移动到新对象之后。另外,使用 set()方法,我们可以用新的对象替换索引位置上的对象。

```
void add(int index,Object element)
boolean addAll(int index,Collection c)
Object set(int index,Object element)
```

我们也可以使用以下三个新方法从一个容器中获取对象。前两个方法返回一个新的迭代器称为 ListIterator,该迭代器提供了双向访问。第一个方法返回全部集成的一个 ListIterator,第二个方法只返回集成中从指定索引位置开始的一个迭代器。第三个方法允许检索指定索引

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

位置上的对象。

```
ListIterator listIterator( )
ListIterator listIterator(int index)
Object get(int index)
```

显然,我们也希望能够以新的方法从集成中删除对象。第一个新方法从指定的索引位置删除元素,而第二个方法删除从指定的索引位置 fromIndex 位置(不包括该位置)开始到索引位置 toIndex 的所有对象。

```
Object remove(int index)
void removeRange(int fromIndex,int toIndex)
```

最后,了解一个元素在集成中的索引位置通常也是很有用的。用来获取元素的三个方法将返回元素所存储的索引位置,或者在元素没有找到的情况下返回 -1。第一个方法从指定的索引位置 startingIndex 开始查找,返回所找到的第一个匹配对象的索引位置;第二个方法只返回可以找到的对象中的最后一个索引;最后一个方法返回最后一个索引,但该索引值不得小于指定的 minIndex。

```
int indexOf(Object o,int startingIndex)
int lastIndexOf(Object o)
int lastIndexOf(Object o,int minIndex)
```

46.2.3 Map 接口

第二个从集成中扩展的接口是 Map(映射)接口。与 List(列表)接口不同,映射接口中保证集成的一个对象至多只有一个实例并且至多只有一个 null(空)。映射接口包含一个元素和一个键字值;键字值决定了元素在映射接口中应该放置的位置,在映射接口中键字不能重复。

向映射中插入元素时必须同时提供键字和值。所以显然应该有一种新的方法来完成这一工作(put( ))。另外,由于映射需要知道键字,所以只拷贝集成并向其中插入是不可能的。相反,putAll( )方法使用与每个元素相关联的键字,拷贝每一个元素。

```
Object put(Object Key,Object value)
Void putAll(Map t)
```

为了从一个映射中获取对象,一般来说我们不必知道对象本身,只要知道其键字即可。get( )方法返回与键字相匹配的元素。另外,映射可以返回一个新的接口称为集合(Set)。有两个方法返回所有项的集合或所有键字。最后一个从集成中获取元素的方法返回映射中有规则的元素的集成。

```
Object get(Object key)
Set entrySet( )
Set keySet( )
Collection values( )
```

与向映射中插入新元素一样,删除元素也是通过键字值,而不是通过元素本身实现的,方法为 remove( )。

```
Object remove(Object key)
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

映射中最后两个新的方法用来确定是否能找到指定的键字或元素。如果映射中存在指定的键字，containsKey()方法返回 true(真)，而如果指定的对象通过一个或多个键字映射到接口中，containsValue()返回 true(真)。

```
boolean containsKey(Object key)
boolean containsValue(Object value)
```

46.2.4　Iterator 接口

正如我们所看到的，集成允许我们通过一个称为 Iterator(迭代器)接口的新接口观察其内容。如果对从 Java API 1.0 开始就已经有了的枚举(Enumeration)接口熟悉的话，那么对迭代器就不应该感到陌生。然而迭代器也允许从底层集成中删除元素。

第一个方法可使我们知道迭代器接口中是否还有任何元素。

```
boolean hasNext( )
```

第二个方法将返回迭代器中的下一个元素。

```
Object next( )
```

最后一个方法可以用来从底层集成中删除迭代器中所读出的最后一个元素。

```
void remove( )
```

46.2.5　ListIterator 接口

列表(List)具有提供更专用的迭代器(Iterator)形式的功能。列表迭代器(ListIterator)利用了列表(List)中的索引，允许用户执行一些其它的操作。

第一个新的功能是插入新对象，或修改刚刚所读取的对象的值。add( )方法用来插入一个新对象，而 set ()方法使用一个指定的对象来替换刚刚从迭代器中读出的对象。

```
void add(Object o)
void set(Object o)
```

除了迭代器中允许读取下一个元素的功能之外，列表迭代器还允许用户读取前一个元素。列表迭代器接口中除了 hasNext()方法之外，增加了 hasPrevious()方法。

```
boolean hasPrevious( )
```

显然，我们也希望能够提取前一个对象，所以列表迭代器接口中除了 next()方法之外，还有一个 previous()方法。

```
Object previous()
```

最后，由于我们所查找的是一个列表，所以也可以获得下一个或前一个元素的索引值。

```
int nextIndex( )
int previousIndex( )
```

46.3　Vector 类

Java 的数组具有很强的功能，但它并不总是能满足我们的要求。有时候我们需要将一些

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

项放到一个数组中,但我们又不能确切地知道有多少项。解决这一问题的一种办法是,创建一个比我们想象的所需要的数组还要大的数组。这种方法正是昔日使用 C 语言编程时所采用的典型方法。对这一问题向量(Vector)对象给出了另外一种解决办法。一个向量与一个数组很相似,向量中也有多个对象,并且我们也可以使用索引值提取对象。向量与数组之间的最大区别是当使用向量超出原有的空间时它会自动增加空间。另外向量还提供了增加和删除元素的额外方法,而这些操作在数组中一般来说必须手工完成。例如,在两个元素之间插入一个元素。向量更有效的一点是它是一个可扩充的数组。

在 Java 2 API 之前,向量只扩展对象(Object)。现在,Java 2 API 中,向量扩展了抽象列表(AbstractList),抽象列表实现了列表(List)。

46.3.1 创建向量

在建立向量时,我们可以指定向量的初始大小及其可能的增长速度。我们也可以只指定向量的初始大小,而让其自己判断它的增长速度,或者我们让向量自己来确定一切。为了实现这几种形式的初始化,向量类提供了三个构造器。

```
public Vector( )
```

该方法创建一个空的向量。

```
public Vector(int initialCapacity)
```

该方法创建一个具有 initialCapacity 个元素空间的向量。

```
public Vector(int initialCapacity,int capacityIncrement)
```

该方法创建一个具有 initialCapacity 个元素空间的向量,当向量需要增长时,增加 capacityIncrement 个元素。

JDK 1.2 中增加了一个构造器用来支持集成(Collection)API,我们将在本章的末尾处讨论。新的构造器创建了一个初始状态有集成中所有元素的向量,其顺序为集成的迭代器所显示的顺序。

如果我们对一般的情况下所增加的元素个数有具体要求,可以事先使用所需元素个数的空间设置向量,如果未使用完所有空间,那也没有关系,我们只要不让向量一次一次地分配更多的空间即可。

警告:如果我们未指定容量增量,则向量在需要增长时,其空间增加一倍。如果我们的向量很大,则这将不是我们所希望看到的行为。当向一个向量添加许多元素时,我们应该设置一个指定的容量增量。即使是由于向量的增长开销很大,我们不进行大量元素的添加,那么如果我们很幸运,知道向量中将会有多少个对象,我们应该在创建向量时使用这一初始容量。

46.3.2 向向量中添加对象

除了标准列表接口(List)中的方法,向一个向量中添加新对象还有两种方式。我们可以作为向量中的最后一个元素添加一个对象,也可以在两个已存在的对象之间插入一个对象。addElement 方法以最后一个元素的方式添加一个对象:

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
public final synchronized void addElement(Object newElement)
```

insertElementAt()方法在指定的位置添加一个新对象。参数 Index 表示向量中新对象应该放置的位置：

```
public final synchronized void insertElementAt(Object newElement,
    int index) throws ArrayIndexOutOfBoundsException
```

如果我们试图在一个并不存在的位置插入新对象，例中，如果我们试图在 9 这个位置上插入新对象，而向量中只有 5 个元素，则我们将获得一个 ArrayIndexOutOfBoundsException 异常。

使用 insertElementAt 方法，我们可以修改向量中指定位置上的对象：

```
public final synchronized void setElementAt(Object ob, int index)
    throws ArrayOutOfBoundsException
```

该方法的工作方式几乎与 insertElementAt 方法完全一样，只是在该方法中向量中的其它元素不必移动位置为新对象留出空间。换句话说，新对象替换了向量中原有的对象。

46.3.3　访问向量中的对象

不幸的是，访问向量中的对象不像访问数组中的元素那么简单。我们不是使用将给定的索引值用方括号([ ])括起来的方法，而是使用 elementAt 方法来访问向量中的元素。向量中等价于 someArray[4] 的是 someVector. elementAt(4)。elementAt 方法的格式为：

```
public final synchronized Object elementAt(int index)
    throws ArrayIndexOutOfBoundsException
```

使用 firstElement 和 lastElement 方法，我们也可以访问向量中的第一个和最后一个元素：

```
public final synchronized Object firstElement()
    throws NoSuchElementException
public final synchronized Object lastElement()
    throws NoSuchElementException
```

如果向量中没有对象，则这两个方法均产生一个 NoSuchElementException 异常。

使用 isEmpty 方法，我们可以测试向量中是否有元素：

```
public final boolean isEmpty( )
```

很多情况下，我们需要使用向量建立一个对象容器，然后将向量转换为 Java 的数组，以提高速度。一般情况下只有当已经得到所需的所有对象后才这样做。例如，如果我们从一个包含许多对象的文件中读取对象，并将这些对象存储在一个向量中，当已经读完文件时，我们创建一个对象数组，并将向量中的对象拷贝到数组中，size 方法可以告诉我们向量中所存放的对象个数：

```
public final int size()
```

当我们知道了向量的大小之后，使用该大小值创建一个对象数组。向量类提供了将向量中所有对象拷贝到对象数组中的一个很方便的方法：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public final synchronized void copyInto(Object[] obArray)
```

如果我们试图拷贝到数组中的对象个数比数组本身可容纳的个数多，则会产生一个 ArrayIndexOutOfBounds 异常。下面的代码段创建了一个对象数组，并将一个称为 myVector 的向量的内容拷贝到该数组：

```
Object obArray[] = new Object[myVector.size()];//创建对象数组
myVector.copyInto(obArray);//拷贝向量到数组
```

46.3.4 Enumeration 接口

如果我们要循环使用一个向量中的所有元素，我们可以使用元素方法获得该向量的一个 Enumeration(枚举)对象。枚举主要用于连续地访问一个数据结构中的元素的情况。它包含两个方法。

```
public abstract boolean hasMoreElements()
```

当其中仍有元素可以访问时该方法返回 true(真)。当其中已经没有元素时，该方法返回 false(假)。

```
public abstract Object nextElement( )
    throws NoSuchElementException
```

该方法返回数据结构中下一个元素的一个引用。如果其中已经没有元素可以访问，而我们调用了该方法，则我们会得到一个 NoSuchElementException 异常。

在向量类的情况下，elements( )方法为向量返回一个枚举接口：

```
public final synchronized Enumeration elements( )
```

下述的代码段使用枚举接口检验一个向量中的所有对象：

```
Enumeration vectEnum = myVector.element( );//获得向量的枚举
while(vectEnum.hasMoreElement( )) //当仍可获得元素时 ...
{
    Object nextOb = vectEnum.nextElement( );//获得下一个元素
    //使用下一个元素进行所需的操作
}
```

该循环对每一个能够返回一个枚举对象的数据结构以同样的方式工作。每个数据结构一般都有一个 elements()方法，或有一个类似的方法，用来返回枚举接口。在这之后，数据结构的类型就无关紧要了——通过枚举接口它们看上去都是一样的。

46.3.5 在向量中搜索对象

我们总是可以通过使用枚举接口逐个元素地进行比较，在一个向量中手工地搜索对象，但是使用内置的搜索函数将会节省大量的时间。

如果我们只是需要知道向量中是否存在某一个对象，则可以使用 contains()方法。例如：

```
public final boolean contains(Object ob)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果指定对象 ob 在向量中至少出现一次，则该方法返回 true(真)，否则该方法返回 false(假)。

我们还可以使用 indexOf( )和 lastIndexOf( )方法，找出对象在向量中的位置。例如：

```
public final int indexOf(Object ob)
```

该方法返回指定对象 ob 在向量中第一次出现的位置，如果 ob 在向量中不存在则返回 - 1。

```
public final synchronized int indexOf(Object ob,int startIndex)
    throws ArrayIndexOutOfBoundsException
```

该方法返回从指定位置 startIndex 开始，指定对象 ob 在向量中第一次出现的位置。如果没有找到，则返回 - 1。如果 startIndex 小于 0，或大于等于向量的长度，则获得一个 ArrayOutOfBoundsException 异常。

```
public final int lastIndexOf(Object ob)
```

该方法返回指定对象 ob 在向量中最后一次出现的位置，如果 ob 在向量中存在，则返回 - 1。

```
public final synchronized int lastIndexOf(Object ob, int startIndex)
    throws ArrayOutOfBoundsException
```

该方法返回从指定位置 startIndex 开始，指定对象 ob 在向量中最后一次出现的位置。如果没有找到 ob 则返回 - 1。如果 startIndex 小于 0，或大于等于向量的长度，则获得一个 ArrayOutOfBoundsException 异常。

46.3.6　从向量中删除对象

从向量中删除对象有三种方法。我们可以删除全部对象，可以删除某一指定的对象，也可以删除某一指定位置上的对象。removeAllElements 方法删除一个向量中的所有对象：

```
public final synchronized void removeAllElements( )
```

removeElement 方法从向量中删除一个指定对象：

```
public final synchronized void removeElements(Object ob)
```

如果对象在向量中多次出现，那么该方法仅删除第一次出现的对象。如果确实删除了指定的对象，则该方法返回 true(真)，如果所要删除的对象在向量中未找到，则该方法返回 false(假)。

removeElementsAt 方法删除向量中指定位置上的对象，并把其它对象进行移动，填充到由于删除对象而空出的空间：

```
public final synchronized void removeElementsAt(int index)
    throws ArrayIndexOutOfBoundsException
```

如果我们试图从一个不存在的位置上删除对象，则将会获得一个 ArrayIndexOutOfBoundsException 异常。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

46.3.7 改变向量的大小

向量有两个大小概念——一个是存储在向量中的元素的个数,另一个是向量的最大容量。capacity 方法可以告诉我们一个向量在必须增长之前可以容纳的对象个数:

```
public final int capacity( )
```

使用 ensureCapacity 方法可以增加向量的容量。例如:

```
public final synchronized void ensureCapacity(int minimumCapacity)
```

该方法通知向量至少应该能够存储 minimumCapacity 个元素。如果向量的当前容量小于 minimumCapacity,则该方法将为向量分配更多的空间。如果向量的当前容量已经大于 minimumCapacity,向量的当前容量不会缩减。

如果我们要减小一个向量的容量,可使用 trimToSize 方法:

```
public final synchronized void trimToSize( )
```

该方法将一个向量的容量减小到其当前所存储的元素个数。

size 方法返回向量中当前所存储的元素个数:

```
public final int size( )
```

使用 setSize 方法,可以修改当前元素的个数:

```
public final synchronized void setSize(int newSize)
```

如果新的大小比原来的大小小,则向量尾部的元素将丢失。如果新的大小比原来的大小大,则新的元素设置为 null。调用 setSize(0)与调用 removeAllElements( )的结果是相同的。

46.4 HashTable 类

HashTable(散列表)类是映射集成最常见的实现,它提供了将一个对象与另一个对象相关联的方法。散列表经常把一个名字与一个对象相关联,并根据该名字提取对象。在一个字典中名字对象称为键字(key),它可以是任何类型的对象。与键字相关联的对象称为值。一个键字只能与一个值相关联,但一个值可以与多个键字相关联。

我们可以将散列表类想象成为一个使用键字对象的散列码来执行查寻的类。它根据散列码将键字分组为桶(bucket)。当要查找一个键字时,首先查找键字的散列码,再使用散列码得到正确的桶,最后在桶中找出正确的键字。一般情况下,一个桶中的键字个数要比散列表中键字的总数少,所以在散列表中所执行的比较只是在大多数集成如向量中所执行的比较的一部分。

散列表有一个容量,该容量指出它所使用的桶数。散列表还有一个加载因子,它是表中元素的个数与桶的个数的比。当创建一个散列表时,我们可以指定一个加载因子门限值。当前加载因子超过这一门限值时,表的容量增加——即它将桶的个数增加一倍并重新组织表。缺省的加载因子门限值为 0.75,即当表中所存储的元素个数是桶数的 75%时,桶的个数增加一倍。我们可以指定任何一个大于 0 小于等于 1 的数作为加载因子门限值。门限值越小,查找速度越快,因为每个桶中将只有几个键字(也许一个都没有),但是整个表中桶的个数将远远多

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

于元素的个数，造成空间浪费。较大的门限值意味着查找速度可能会较慢，但是桶的个数会接近于元素的个数。

散列表(Hashtable)类有三个构造器：

```
public Hashtable()
```

该构造器使用缺省容量 101 和缺省加载因子门限值 0.75，创建一个散列表。

```
public Hashtable(int initialCapacity)
```

该构造器使用指定的初始容量和缺省的加载因子门限值 0.75，创建一个散列表。

```
public Hashtable(int initialCapacity, float loadFactorThreshhold)
    throws IllegalArgumentException
```

该构造器使用指定的初始容量和门限值创建一个新的散列表。如果所指定的初始容量值小于等于 0，或者如果所指定的门限值小于等于 0 或大于 1，则将获得一个 IllegalArgumentException 异常。

46.4.1　散列表中存储对象

要使用指定键字在字典中存储一个对象，可使用 put()方法：

```
public abstract Object put(Object key,Object value)
    throws NullPointerException
```

put 方法所返回的对象是先前与该键字相关联的对象。如果不存在先前与该键字相关联的对象，该方法返回 null。不允许使用 null 键字或 null 值。如果传送了 null 键字或 null 值，则该方法返回一个 NullPointerException 异常。

46.4.2　从散列表中检索对象

使用 get( )方法可以查找散列表中与指定键字相关联的对象：

```
public abstract Object get(Object key)
```

如果不存在与指定键字相关联的值，get()方法返回 null。

46.4.3　从散列表中删除对象

为从字典中删除一个键字——值对，可使用该键字调用 remove()方法。例如：

```
public abstract Object remove(Object key)
```

该方法返回与指定键字相关联的对象，如果不存在与该键字相关联的值，该方法返回 null。

字典(Dictionary)类也提供了一些实用的方法，为用户提供有关字典的信息。如果字典中没有存放任何对象，则 isEmpty()方法返回 true(真)：

```
public abstract boolean isEmpty( )
```

size 方法则返回字典中当前所存储的键字——值对的个数：

```
public abstract int size( )
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

keys 方法返回一个枚举(Enumeration)对象,允许用户检查字典中的所有键字,而 elements()方法则返回字典中所有值的一个枚举:

```
public abstract Enumeration keys( )
public abstract Enumeration elements( )
```

除了这些方法之外,散列表还有一些关于映射接口的方法。

```
public synchronized void clear( )
```

该方法从一个散列表中删除所有元素。该方法与向量(Vector)类中的 removeAllElements 方法类似。

```
public synchronized boolean contains(Object value)
    throws NullPointerException
```

如果散列表中存在所指定的值 value,该方法返回 true(真);如果指定的值是 null,该方法返回一个 NullPointerException 异常。

```
public synchronized boolean containsKey(Object key)
```

如果指定的键字 key 作为键字在散列表中存在,该方法返回 true(真)。

当散列表的大小增加时,散列表必须在新的桶集合上重新安排表中的所有对象。换句话说,如果表中原有 512 个桶,增长到 1024 个桶,则我们必须在全部 1024 个桶上重新分布对象。一个对象的桶是由散列码和桶数共同决定的。如果我们改变了桶数,但未重新排列对象,由于对象的桶仍基于一个较小的尺寸,散列表可能无法定位一个已有的对象。rehash( ) 方法(当表增长时自动调用)重新计算每个对象在表中的位置。

46.5 Properties 类

Properties(属性)类是一个使用键字和值字符串的特殊种类的字典。系统(System)类使用它可以存储系统属性,我们使用它可以创建一套自己的属性。属性类实际上只是一个专门用来存储字符串的散列表。

使用下面的空构造器我们可以创建一个新的属性对象:

```
public Properties( )
```

我们也可以使用一组缺省的属性创建一个属性对象。当属性对象不能在其自己的表中找到一个属性时,它将搜索缺省的属性表。如果我们修改自己的属性对象中的一个属性,这并不影响缺省属性对象中的相应属性。这意味着多个属性对象可以安全地共享同一个缺省的属性对象。为了使用一组缺省属性创建一个属性对象,只要将缺省属性对象传送给构造器即可:

```
public Properties(Properties defaultProps)
```

46.5.1 设置属性

使用所有字典都使用的同一方法 put( ),我们可以设置属性:

```
public Object put(Object key, Object value)
```

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
throws NullPointerException
```

46.5.2　查询属性

getProperty()方法返回对应于一个属性名的字符串，如果没有设置该属性，则返回 null：

```
public String getProperty(String key)
```

如果我们指定了一个缺省属性对象，在返回 null 之前也要检查该对象。我们在调用 getProperty( )方法时，也可以指定没有找到属性设置时的缺省返回值：

```
public String getProperty(String key,String defaultValue)
```

在这一版本的 getProperty 方法中，完全不考虑缺省属性对象的情况，所返回的值或者是对应于指定键字的属性，或者是该属性未设置时的 defaultValue。

警告：由于属性类使用了字典类中的 put()方法，我们也可以在一个属性对象中存储字符串以外的对象。然而，如果我们存储一个属性时不是一个 String 或 String 的子类，那么当试图使用 getProperty()方法检索该属性时，将获得一个 ClassCastException 异常。所以，在一个对象中使用 toString()方法以保证所存储的是一个字符串而并非一个非字符串对象，是一个很好的方法。

使用 propertyNames()方法，我们可以获得一个属性对象中所有属性名的一个枚举对象，包括缺省属性：

```
public Enumeration propertyNames( )
```

46.5.3　保存和提取属性

由于属性对于保存像用户的引用这样的信息非常有用，所以我们需要有一种保存属性到一个文件，并在下次程序启动时可以将其读出的方法。对于这一点，我们可以使用 load( )和 save()方法。

```
public synchronized void save(OutputStream out,String header)
```

该方法将属性保存到输出流 out 中，头字符串先于属性对象的内容写到流中。

```
public synchronized void load(InputStream in)
    throws IOException
```

该方法从输入流中读取属性。该方法将 # 和！作为注释字符处理，并忽略掉这种字符后直到行尾的内容，类似于 Java 中的//注释字符。

程序清单 46.1 给出了一个使用 save()方法写入的简单文件。

```
# Example Properties
# Mon Jun 17 19:57:39 1996
foo = bar
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
favoriteStooge = curly
helloMessage = hello world!
```

list()方法类似于 save()方法,但 list()方法以更可读的形式给出属性。它以一种美观、友好的格式在一个打印流上显示出属性表的内容,方便了调试。list()方法的格式如下:

```
public void list(PrintStream out)
```

46.6 Stack 类

Stack(堆栈)是一种很方便的数据结构,它以后进先出的方式添加数据项。换句话说,当我们请求堆栈给出下一个数据项时,它给出的是我们最新添加进去的那个数据项。让我们将堆栈想象成由咖啡碟子摞起来的栈。堆栈顶上的碟子是我们最后放入栈的那一个碟子。当我们每次放入另一个碟子时它都变成了一个新的栈顶。

堆栈类是向量类的一个子类实现的,这就是说除了堆栈特有的方法之外,所有向量的方法均可使用。使用空构造器可创建一个堆栈:

```
public Stack()
```

要向堆栈的顶部增加一个项,我们要将其压入堆栈:

```
public Object push(Object newItem)
```

push()方法所返回的对象是与 newItem 相同的对象。pop()方法弹出堆栈的顶项:

```
public Object pop( ) throws EmptyStackException
```

如果试图从一个空堆栈中弹出一个项,则将获得一个 EmptyStackException 异常。使用 peek()方法,我们可以知道堆栈的顶项是什么项而不弹出:

```
public Object peek( ) throws EmptyStackException
```

如果堆栈中不存在任何项,empty()方法返回 true(真):

```
public boolean empty()
```

有时候我们可能会需要找出一个对象相对于栈顶所处的位置。由于我们并不确切知道堆栈是如何存储数据项的。向量类中的 indexOf()和 lastIndexOf()方法可能并不会对我们有太大帮助。然而 search()方法可以告诉我们一个对象距离栈顶到底有多远:

```
public int search(Object ob)
```

如果所指定的对象根本就不在栈中,search()方法返回 -1。

程序清单 46.2 给出的代码段创建了一个字符串数组,然后使用一个堆栈,通过将其全部压入堆栈,再反向弹出的方法,翻转单词的顺序。

```
String myArray[] = {"Please", "Reverse", "These", "Words"};
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Stack myStack = new Stack();

// Push all the elements in the array onto the stack

for (int i=0; i < myArray.length; i++) {
    myStack.push(myArray[i]);
}
// Pop the elements off the stack and put them in the
// array starting at the beginning

for (int i=0; i < myArray.length; i++) {
    myArray[i] = (String) myStack.pop();
}

// At this point, the words in myArray will be in
// the order: Words, These, Reverse, Please
```

46.7　Date 类

Date(日期)类用来表示指定的日期和时间。它以新纪元 Epoch 为中心,这个新纪元为 1970 年 1 月 1 日格林威治时间(GMT)0 点。虽然在日期类中有一些方法支持早到 1900 年的日期引用,但没有一个日期方法对于新纪元(Epoch)之前的日期正确地起作用。

日期类中的空构造器从当前时间创建一个日期对象:

```
public Date( )
```

我们也可以使用从新纪元开始的毫秒数来创建一个日期对象,这个毫秒数与 System.currentTimeMillis()方法所返回的值相同:

```
public Date(long millis)
```

我们也可以使用日期类中的静态方法 UTC(Universal Time Coordinates) 来获取自新纪元以来的毫秒数:

```
public static long UTC(int year,int month,int date,
    int hours,int minutes,int seconds)
```

下述的日期构造器允许我们通过指定年、月、日等来创建日期对象:

```
public Date(int year,int month,int date)
public Date(int year,int month,int date,int hours,int minutes)
public Date(int year,int month,int date,int hours,int minutes,int seconds)
```

当使用这种方式创建日期对象时,有几个重要的问题需要注意:

- year 的值是 1900 年以来的年数,例如,对 1984 年该值为 84。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 月数从 0 开始，而不是从 1 开始，1 月份为 0。
- 日期（一个月中的某一日）从 1 开始，所以一个月中的第 11 天其日期为 11。
- 时、分、秒均从 0 开始，与月数不同，这种情况是正确的，小时值 1 表示 1a.m（上午 1 点）。

日期类也具有使用字符串表示的日期创建新日期对象的功能：

```
public Date(String s)
```

下述语句均为 1992 年 1 月 12 日（HAL9000 计算机的诞生日）创建的日期对象：

```
Date d = new Date("January 12,1992");
Date d = new Date(92,0,12)
Date d = new Date(6951744000001);//自新纪元算起的毫秒数
Date d = new Date(Date.UTC(92,0,12,0,0,0));
```

注意：当使用指定的年、月、日、时、分、秒值创建日期时，或当打印出一个日期对象的值时，都使用的是本地时区。UTC 方法和自新纪元算起的毫秒数则是使用格林威治（GMT，Greenwich Mean Time）时间。

46.7.1 比较日期

与所有 Object 的子类一样，我们可以使用 equals()方法比较两个日期。日期类也提供了用来确定一个日期是在另一个日期之前或之后的方法。如果对象日期在传送给方法的日期之后，则 after()方法返回 true（真）：

```
public boolean after(Date when)
```

假设我们定义了两个日期 date1 和 date2，如下所示：

```
Date date1 = new Date(76,6,4);//1976 年 7 月 4 日
Date date2 = new Date(92,0,12);//1992 年 1 月 12 日
```

对于这两个日期，date1.before(date2)返回 true（真），date1.after(date2)返回 false（假）。

46.7.2 转换日期为字符串

我们总是可以使用 toString( )方法将日期转换为一个字符串。它将日期转换为一个使用本地时区表示的字符串。toLocaleString 方法也将日期转换为一个使用本地时区表示的字符串，但字符串的格式稍有不同：

```
public String toLocaleString()
```

toGMTString( )方法使用格林尼治时间作为时区，将一个日期转换为一个字符串：

```
public String toGMTString()
```

下面的例子给出了不同字符串转换的格式。原始时间定为格林尼治时间 1992 年 1 月 12 日 0 点。时区为东部标准时间，或比格林尼治时间晚五个小时。

```
Sat Jan 11 19:00:00 1992 //toString
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
01/11/92 19:00:00 //toLocaleString
12 Jan 1992 00:00:00 GMT //toGMTString
```

46.7.3　改变日期属性

我们可以查询和修改日期的几乎所有部分。只有两个部分只能查询不能修改，这就是时差和日期对象的星期数。时差为当地时区与格林尼治时间之间相差的分钟数。如果当地时间晚于格林尼治时间，时差为正数，即如果格林尼治时间的午夜早于当地时区的午夜，则时差为正。getTimezoneOffset()方法的格式为：

```
public int getTimezoneOffset()
```

getDay( )方法返回一个 0 到 6 之间的数，此处 0 表示星期日：

```
public int getDay()
```

切记，星期的计算使用当地时间。

如果我们喜欢以自新纪元的毫秒数处理日期，可使用 getTime( )和 setTime( )方法来修改日期：

```
public long getTime( )
public void setTime(long time)
```

使用下面的方法我们还可以对日期的单个部分进行操作：

```
public int getYear( )
public int getMonth( )
public int getDate( )
public int getHours( )
public int getMinutes( )
public int getSeconds( )
public void setYear(int year)
public void setMonth(int month)
public void setDate(int date)
public void setHours(int hours)
public void setMinutes(int minutes)
public void setSeconds(int seconds)
```

46.8　BitSet 类

BitSet(位集)类为进行多个位或单位操作提供了方便的方法。BitSet 类在处理更多的位时自动地增长。使用空构造器我们可以创建一个空位集(bit set)：

```
public BitSet( )
```

如果我们知道所需要的位数，应该使用指定的尺寸创建 BitSet：

```
public BitSet(int numberOfBits)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

位类似于灯的开关,可以打开或关闭。如果某一位被设置,则认为它是打开的,相反,如果清除某位则认为它是关闭的。位经常与布尔(Boolean)值联系在一起,因为它们都只有两个可能的值。一个被设置的位认为是 true(真),相反一个被清除的位认为是 false(假)。

在位集中我们使用 set( )和 clear( )方法来设置和清除一个单一的值:

```
public void set(int whichBit)
public void clear(int whichBit)
```

如果我们创建一个具有 200 个位的位集,并且试图设置第 438 位,位集将自动地增加到至少包含 438 个位。新增加的位全部初始化为清除状态。size ( )方法可返回当前位集中的位数:

```
public int size( )
```

使用 get( )方法我们可以测试一个位是设置还是清除状态:

```
public boolean get(int whichBit)
```

如果指定位被设置,get()方法返回 true(真),如果指定位被清除则返回 false(假)。

在两个位集之间可以进行以下三种操作。这些操作使用来自于第二个位集中的位操作当前位集。以相应的位进行匹配来执行操作,也就是说,当前位集中的 0 位与第二个位集中的第 0 位进行比较。这些位操作为:

or(或)操作:如果当前位或第二个位集中的对应位设置,则设置当前位集中的该位。否则,清除当前位。

and(与)操作:如果当前位与第二个位集中的对应位均设置,则设置当前位集中的该位。否则,清除当前位。

xor(异或)操作:如果相应的两个位中只有一个位被设置,则设置当前位集中的该位。否则,清除该位。

这些位操作的格式为:

```
public void or(Bitset bits)
public void and(Bitset bits)
public void xor(Bitset bits)
```

46.9 StringTokenizer 类

StringTokenizer 类通过将字符串分解成标识的方法,帮助我们对字符串进行分析。它根据一个分隔符集来识别标识。一个标识可视为由非分隔符字符组成的字符串。例如,短语"I am a sentence"对空格分隔符包含几个标识,这些标识为 I、am、a、sentence。

如果我们使用冒号作为分隔符,则整个句子将是一个长标识"I am a sentence",因为其中不含有冒号将其分隔开。StringTokenizer 类并非一定要使用空格进行单词分隔。如果我们指定只使用冒号分隔单词,则空格将被看作为单词的一部分。

我们可以使用一个分格符集,也就是可使用许多不同的字符来分隔标识。例如,如果我们有一个串"Hello. How are you? I am fine, I think,"我们可能要使用空格、句点、逗号、问号作为分隔符,将句子分解成只有单词的标识。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

字符串标识化工具本身并没有单词的概念，它只识别分隔符。我们在分析文本时，通常使用空白符作为分隔符。空白符包括空格、制表符、换行符及回车符。我们在创建一个字符串标识化工具时，如果不指定分隔符串，则将使用空白符进行分隔。

把需要被标识化的字符串传送给构造器，可以创建一个字符串标识化工具：

```
public StringTokenizer(String str)
```

如果我们需要使用空白符以外的字符作为分隔符，还可以传送一个包含分隔符的字符串：

```
public StringTokenizer(String str,String delimiters)
```

有时候我们需要知道用来分隔两个标识的分隔符是什么。我们可以使用下面的构造器，在 returnTokens 参数中传送 true(真)，要求字符串标识化工具将分隔符作为标识返回：

```
public StringTokenizer(String str,String delimiters,boolean returnTokens)
```

nextToken()方法返回字符串中的下一个标识：

```
public String nextToken( ) throws NoSuchElementException
```

如果字符串中已经没有下一个标识，则产生一个 NoSuchElementException 异常。在使用 nextToken()方法之前，我们可以使用 hasMoreToken( )方法确定是否下面还有标识：

```
public boolean hasMoreTokens( )
```

我们也可以向 nextToken()方法传送一个新的分隔符集，临时改变所使用的分隔符集：

```
public String nextToken(String newDelimiters)
```

新的分隔符集在下一个标识分隔出之前生效，直到它们再次被改变。

countTokens( )方法返回分隔符集不改变的情况下，字符串中的标识个数：

```
public int countTokens( )
```

读者或许已经注意到，nextToken( )和 hasMoreTokens ()方法与枚举接口中的 nextElements()方法很相似。事实上它们确实很相似，所以 StringTokenizer 也实现了一个枚举接口。如下所示：

```
public boolean hasMoreElements( ){
    return hasMoreTokens( );
}
public Object nextElement( ){
    return nextToken( );
}
```

下面的代码段使用了一个字符串标识化工具，打印出一个句子中的单词：

```
String sentence = "This is a sentence";
StringTodenizer tokenizer = new StringTokenizer(sentence);
while(tokenizer.hasMoreTokens())
{
    System.out.println(tokenizer.nextToken());
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

46.10 Random 类

Random(随机数)类提供了一个比数学(Math)类中的随机数发生器更灵活的随机数发生器。实际上,数学类中的随机数发生器只使用了随机数类中的一种方法。由于随机数类中的方法为非静态的,所以我们在产生随机数之前必须创建一个随机数实例。完成这一工作最简单的方法就是使用空构造器:

```
public Random( )
```

随机数类一个方便的特性是,它允许用户设置随机数种子,以确定随机数的模式。虽然我们并不能根据特定的种子方便地预测出所产生的将是一些什么数,但是使用相同的种子我们可以重复一系列的随机数。换句话说,如果我们每次使用同一个随机数种子创建一个随机数实例,我们每次都将获得相同的随机数序列。这对于编写游戏来说并不好,并且对于彩票来说在财政上将是致命的,但这对于编写需要一次又一次地重复同一过程的仿真程序却非常有用。

空构造器使用 System.CurrentTimeMillis 来生成随机数发生器。为了使用一个特殊的种子创建随机数实例,只需将该种子值传送给构造器即可。

```
public Random(long seed)
```

使用 setSeed( )方法我们可以随时修改随机数发生器的种子:

```
public synchronized void setSeed(long newSeed)
```

随机数类可以以四种不同的数据类型产生随机数。

```
public int nextInt( )
```

该方法产生一个 32 位的合法整型值随机数。

```
public long nextLong( )
```

该方法产生一个 64 位的合法长整型值随机数。

```
public float nextFloat( )
```

该方法产生一个 0.0 到 1.0 之间且总是小于 1.0 的浮点随机数。

```
public double nextDouble( )
```

该方法产生一个 0.0 到 1.0 之间且总是小于 1.0 的双精度浮点随机数。这正是 Math.random( )所使用的方法。

还有一种具有某些有趣的数学性质的随机数的特殊变型,这种变型称为 nextGaussian。

```
public synchronized double nextGaussian( )
```

该方法产生一种特殊的合法双精度随机数。该方法所产生的随机数的平均值为 0.0,标准偏差为 1.0。这意味着该方法所产生的随机数通常接近于零,并且很少有较大的数。

46.11 Observable 类

Observable(可观察)类允许一个对象当其改变时通知其它对象。可观察这一概念借用于

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Smalltalk。在 Smalltalk 中，一个对象可以对另一对象表示兴趣，即它希望能知道其它对象的改变情况。

在建立用户界面时，我们可能有多种方式修改某一个数据片，并且这些数据片的变化有可能引起若干个不同的显示区的更新。例如，假设我们需要创建一个改变整数值的滚动条，而这一整数值又以某种图形标尺的形式显示。当该整数值改变后，我们需要标尺也得到更新，但我们又不希望标尺知道关于滚动条的任何东西。如果我们不知道为什么标尺不应该了解滚动条，我们可以设想将滚动条换成使用文本字段来输入所需要的数值，这时将会怎样？我们不必在每次改变输入源时去修改标尺。

我们最好是创建一个可观察的整型变量，这将允许其它对象对它给予关注。当该整型变量改变时，它将通知那些关注它的对象（称为观察者），它已经改变了。在上述图形标尺的例子中，图型标尺将得到值已经改变的通知，它将查询整型变量的新值，并重新绘制其本身。这样就使得标尺可能正确地显示数据，而不管我们使用什么手段来改变值。

这一概念称为模型视图控制器（Model - View - Controller）。模型是应用程序的非可视部分。在前面的例子中，其模型是一个单个的整型变量。视图是模型某些部分的可视显示。图形标尺就是视图的一个例子，滚动条也是视图的一个例子，因为当整数值改变时，滚动条的位置也随之更新。控制器是任何可修改视图的输入源。在这种情况下，滚动条也是一个控制器（可以同时是一个视图和一个控制器）。

在 Smalltalk 中，表示关注一个对象的这种机制正好建立在 Object 类中。不幸的是，由于某种原因，Sun 公司将这种观察机制分割为一个单独的类。这意味着给用户增加了额外的工作，用户不能只注册所关注的 Integer 类，必须创建自己的可观察子类。

创建可观察子类最重要的方法是 setChanged( ) 和 notifyObservers( )。setChanged( )方法用来标记可观察的对象已经改变，以便使用户可以调用 notifyObservers( )方法通知观察者：

```
protected synchronized void setChanged( )
```

setChanged( )方法设置一个由 notifyObservers( )方法使用的内部 changed 标志。该标志在 notifyObservers( )方法调用时自动被清除，但我们也可以使用 clearChanged( )方法人为清除：

```
protected synchronized void clearChanged( )
```

notifyObservers( )方法检查 changed 标志是否设置，如果没有，该方法不发出任何通知：

```
public void notifyObservers( )
```

下述的代码段设置 changed 标志并通知观察者所发生的改变：

```
setChanged( ); //标志可观察对象改变
notifyObservers( ); //将改变通知观察者
```

也可以使用一个参数来调用 notifyObservers( )方法：

```
public void notifyObservers(Object arg)
```

该参数可用来传送关于变化的一些附加信息例如新的值。不带参数调用 notifyObservers( )方法与使用一个 null 参数调用该方法的结果相同。

调用 hasChanged( )方法，可以确定可观察对象是否已经改变：

```
public synchronized boolean hasChanged( )
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

调用 addObserver( )方法,观察者可以登记其对可观察对象的关注:

```
public synchronized void addObserver(Observer obs)
```

调用 deleteObserver( )方法,观察者可以取消其对可观察对象关注的登记:

```
public synchronized void deleteObserver(Observer obs)
```

一个可观察对象可以通过调用 deleteObservers( )方法,清除其观察者列表:

```
public synchronized void deleteObserver()
```

countObservers( )方法返回登记于一个可观察对象的所有观察者的个数:

```
public synchronized int countObservers()
```

程序清单 46.3 给出了一个可观察整型类 ObservableInt 的实现示例。

程序清单 46.3 ObservableInt 的实现代码

```
import java.util.*;

// ObservableInt - an integer Observable
//
// This class implements the Observable mechanism for
// a simple int variable.
// You can set the value with setValue(int)
// and int getValue() returns the current value.

public class ObservableInt extends Observable
{
    int value;      // The value everyone wants to observe

    public ObservableInt()
    {
        value = 0;      // By default, let value be 0
    }

    public ObservableInt(int newValue)
    {
        value = newValue;      // Allow value to be set when created
    }

    public synchronized void setValue(int newValue)
    {
//
// Check to see that this call is REALLY changing the value
//
        if (newValue != value)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            }
                value = newValue;
                setChanged();      // Mark this class as "changed"
                notifyObservers();      // Tell the observers about it
            }
        }

        public synchronized int getValue()
        {
            return value;
        }
    }
```

可观察类有一个对等的接口称为观察者(Observer)。任何需要接收有关一个可观察对象的修改情况的类,都必须实现观察者接口。观察者接口只包括一个称为 update( )的方法,该方法在对象改变时调用。update( )方法的格式为:

```
public abstract void update(Observable obs,Object arg);
```

其中 obs 是刚刚发生变化的可观察对象,arg 是由可观察对象在调用 notifyObservers()时所传送的值。如果调用 notifyObservers() 时不带参数,arg 为 null。

程序清单 46.4 给出了一个 Label 类示例,该 Label 类实现了观察者接口,它可以使用一个整型变量获得各种变化的通知,并使用新值更新其自己。

程序清单 46.4　IntLabel.java 的源代码

```
import java.awt.*;
import java.util.*;

//
// IntLabel - a Label that displays the value of
// an ObservableInt.

public class IntLabel extends Label implements Observer
{
    private ObservableInt intValue;      // The value we're observing

    public IntLabel(ObservableInt theInt)
    {
        intValue = theInt;

// Tell intValue we're interested in it

        intValue.addObserver(this);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// Initialize the label to the current value of intValue

            setText("" + intValue.getValue());
        }

// Update will be called whenever intValue is changed, so just update
// the label text.

        public void update(Observable obs, Object arg)
        {
            setText("" + intValue.getValue());
        }
}
```

现在我们已经有了一个以 ObservableInt 形式定义的模型对象，和一个 IntLabel 形式的视图，我们可以创建一个控制器——IntScrollbar。程序清单 46.5 给出了 IntScrollbar 的实现。


```
import java.awt.*;
import java.util.*;

//
// IntScrollbar - a Scrollbar that modifies an
// ObservableInt.  This class functions as both a
// "View" of the observable, since the position of
// the scrollbar is changed as the observable's value
// is changed, and it is a "controller," since it also
// sets the value of the observable.
//
// IntScrollbar has the same constructors as Scrollbar,
// except that in each case, there is an additional
// parameter that is the ObservableInt.
// Note:  On the constructor where you pass in the initial
// scrollbar position, the position is ignored.

public class IntScrollbar extends Scrollbar implements Observer
{
    private ObservableInt intValue;

// The bulk of this class is implementing the various
// constructors that are available in the Scrollbar class.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public IntScrollbar(ObservableInt newValue)
    {
        super();        // Call the Scrollbar constructor
        intValue = newValue;
        intValue.addObserver(this);          // Register interest
        setValue(intValue.getValue());       // Change scrollbar position
    }

    public IntScrollbar(ObservableInt newValue, int orientation)
    {
        super(orientation);          // Call the Scrollbar constructor
        intValue = newValue;
        intValue.addObserver(this);          // Register interest
        setValue(intValue.getValue());       // Change scrollbar position
    }

    public IntScrollbar(ObservableInt newValue, int orientation,
        int value, int pageSize, int lowValue, int highValue)
    {
        super(orientation, value, pageSize, lowValue, highValue);
        intValue = newValue;
        intValue.addObserver(this);          // Register interest
        setValue(intValue.getValue());       // Change scrollbar position
    }

// The handleEvent method checks with the parent class (Scrollbar) to see
// if it wants the event, if not, just assumes the scrollbar value has
// changed and updates the observable int with the new position.

    public boolean handleEvent(Event evt)
    {
        if (super.handleEvent(evt))
        {
            return true;      // The Scrollbar class handled it
        }
        intValue.setValue(getValue());       // Update the observable int
        return true;
    }

// update is called whenever the observable int changes its value

    public void update(Observable obs, Object arg)
    {
        setValue(intValue.getValue());
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }
}
```

看上去需要做的工作很多，但是请注意，创建一个修改 ObservableInt 的 IntScrollbar 和一个显示其自己的 IntLabel 的小程序是何等容易。程序清单 46.6 给出了一个使用 IntScrollbar、ObserableInt、IntLabel 的小程序实现。

程序清单 46.6 ObservableApplet1.java 源代码

```
import java.applet.*;
import java.awt.*;

public class ObservableApplet1 extends Applet
{
    ObservableInt myIntValue;

    public void init()
    {

// Create the Observable int to play with

        myIntValue = new ObservableInt(5);
        setLayout(new GridLayout(2, 0));

// Create an IntScrollbar that modifies the observable int

        add(new IntScrollbar(myIntValue,
            Scrollbar.HORIZONTAL,
            0, 10, 0, 100));

// Create an IntLabel that displays the observable int

        add(new IntLabel(myIntValue));
    }
}
```

我们可能已经注意到了，当运行这个小程序时，只要改变滚动条，标号的值就更新，然而标号对滚动条并不了解，滚动条也对标号并不了解。

现在，假设我们还允许从一个 TextField 文本字段来更新值，我们所需做的全部工作就是创建一个修改 ObservableInt 的 TextField 的子类。程序清单 46.7 给出了一个 IntTextField 的实现。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 46.7　IntTextField.java 的源代码

```
import java.awt.*;
import java.util.*;

//
// IntTextField - a TextField that reads in integer values and
// updates an Observable int with the new value.  This class
// is both a "view" of the Observable int, since it displays
// its current value, and a "controller" since it updates the
// value.

public class IntTextField extends TextField implements Observer
{
    private ObservableInt intValue;

    public IntTextField(ObservableInt theInt)
    {
// Initialize the field to the current value, allow 3 input columns

        super("" + theInt.getValue(), 3);
        intValue = theInt;
        intValue.addObserver(this);      // Express interest in value
    }

// The action for the text field is called whenever someone presses "return"
// We'll try to convert the string in the field to an integer, and if
// successful, update the observable int.

    public boolean action(Event evt, Object whatAction)
    {
        Integer intStr;              // to be converted from a string

        try {      // The conversion can throw an exception
            intStr = new Integer(getText());

// If we get here, there was no exception, update the observable

            intValue.setValue(intStr.intValue());
        } catch (Exception oops) {
// We just ignore the exception
        }
        return true;
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// The update action is called whenever the observable int's value changes.
// We just update the text in the field with the new int value

    public void update(Observable obs, Object arg)
    {
        setText("" + intValue.getValue());
    }
}
```

当我们创建了这个类后,可以想一下我们需要向小程序添加多少代码呢?我们只需要加入一行代码(并将 GridLayout 改成三行)即可。程序清单 46.8 给出了一个使用 ObservableInt、IntScrollbar、IntLabel 及一个 IntTextField 的小程序的实现。

```
import java.applet.*;
import java.awt.*;

public class ObservableApplet2 extends Applet
{
    ObservableInt myIntValue;

    public void init()
    {

// Create the Observable int to play with

        myIntValue = new ObservableInt(5);

        setLayout(new GridLayout(3, 0));

// Create an IntScrollbar that modifies the observable int

        add(new IntScrollbar(myIntValue,
            Scrollbar.HORIZONTAL,
            0, 10, 0, 100));

// Create an IntLabel that displays the observable int

        add(new IntLabel(myIntValue));

// Create an IntTextField that displays and updates the observable int
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            add(new IntTextField(myIntValue));
        }
    }
```

再一次重申，修改和显示整数值的组件之间互相并不了解，然而一旦修改了该值，它们都会随新值而更新。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 47 章　java.lang

本章内容：

47.1　Java.lang 软件包
47.2　Object 类
47.3　Class 类
47.4　Package 类
47.5　String 类
47.6　StringBuffer 类
47.7　Thread 类
47.8　ThreadGroup 类
47.9　Throwable 类
47.10　System 类
47.11　Runtime 类和 Process 类
47.12　Math 类
47.13　对象包装类
47.14　Character 类
47.15　Boolean 类
47.16　Number 类
47.17　Integer 类
47.18　Long 类
47.19　Byte 类
47.20　Short 类
47.21　Float 类
47.22　Double 类
47.23　Void 类
47.24　java.math.BigInteger 类
47.25　java.math.BigDecimal 类
47.26　ClassLoader 类
47.27　SecurityManager 类
47.28　Compiler 类

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

47.1　Java.lang 软件包

在所有的 Java API 软件包中，Java.lang 是最重要的一个。它所包括的类为其它的 Java 软件包提供了一个坚实的基础。可以肯定地说，java.lang 软件包是一个不需要其它软件包支持的 Java 软件包。

Java.lang 软件包包括如下的一些类：

- Object 类：Object 类是基类，所有其它类都由它派生而来。如果我们没有明确指出我们的新子类是对某个类的扩展，那么它所扩展的就是 Object。
- Class 类：Class 类表示了一个 Java 类，对于 Java 中所定义的每一个类，都有一个描述它的 Class 类实例。
- ClassLoader 类：ClassLoader 类提供了一种向 Java 运行环境添加新类的办法。
- Compiler 类：该类提供了对 Just－In－Time 编译器的访问，如果有这种编译器的话。
- Math 类：Math 类提供了大量的常用数学函数。
- Number 类：Number 类是 Java 的数值类的基础，这些数值类包括 Double、Float、Integer 及 Long，这些类也称为对象包装类，因为它们为内置的原始数据类型提供了一个对象接口。
- Package 类：Package 类定义了关于一个包的信息，包括版本号、规范标题及软件商等。使用 Package 类，JDK 1.2 的应用程序可以保证使用一个包的正确版本。
- Process 类：该类表示一个由 Runtime 对象启动的外部程序。
- Runtime 类：该类提供了许多与 System 类相同的功能，但它也处理外部程序的运行。
- RuntimePermission 类：该类定义了运行访问等级。更详细的介绍参见第 34 章“Java 安全性深析”。
- String 类：String 类提供了操作 Java 字符串的方法。
- StringBuffer 类：该类用于创建 Java 字符串，特别是其长度经常改变的字符串。
- System 类：该类提供了特殊的系统级的工具。
- Thread 类：该类表示了 Java 程序中一个执行的线程。每一个执行中的程序都可以有多个线程在运行。
- ThreadGroup 类：该类允许将线程相互关联起来。有些线程操作只能通过在同一个 ThreadGroup 中的线程来执行。
- Throwable 类：该类是 Java 异常的基础类。任何由 catch 语句所捕获或由 throws 语句所抛出的对象，必须是 Throwable 类的一个子类。
- SecurityManager 类：该类定义了当前运行环境的安全限制。许多 Java 类使用 SecurityManager 检查一个操作是否允许。
- Boolean、Byte、Character、Double、Float、Integer、Long 及 Short 类：这些类是其本地数据类型的对象包装类。例如，Character 包装 char 数据类型。

除了以上这些类之外，java.lang 软件包还定义了四个接口：

- Cloneable：必须使用可以克隆或拷贝的对象实现。
- Comparable：用于定义对象的顺序。这对于像 java.util.List 这样的类非常有用。
- Runnable：用于与 Thread 类连接。
- Runtime.MemoryAdvice：用于与 Runtime 类连接，为 Runtime 的垃圾收集定义强制

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

等级。

47.2 Object 类

Object 类是 Java 中所有类的基础，Java 的每个类都支持它所定义的方法。

47.2.1 检测对象相等

我们已经知道，== 操作符只告诉我们两个对象是否是相同的对象。这与检测对象中包含的信息是否相同并不是一回事。Object 类中的 equals() 方法可使我们定义一种用来判断两个对象中是否包含相同信息的方法。例如，你和我都拥有同一种汽车型号，但是 myCar == yourCar 并不为真——它们是两个不同的对象。然而，如果我们使用 myCar.equals(yourCar) 方法进行测试，它们可能包含相同的信息。equals()方法的格式为：

```
public boolean equals(Object ob)
```

程序清单 47.1 给出一个使用 equals()方法的样本类，它对两个对象逐条属性地进行比较。

```
public class EqualityTest
{
    protected String someName;
    protected int someNumber;
    protected Object someObject;

    public boolean equals(Object otherOb)
    {
        EqualityTest other;

// First, test to see if these are the same object
        if (otherOb == this) return true;

// Next, make sure the other object is the same class
        if (! (otherOb instanceof EqualityTest)) return false;

// Cast otherOb to this kind of object (EqualityTest) for accessing
// the attributes.
        other = (EqualityTest) otherOb;

// Now, compare each attribute of the objects to see if they are equal.
// Notice that on primitive data types like int you should use ==
        if (someName.equals(other.someName) &&
            (someNumber == other.someNumber) &&
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            (someObject.equals(other.someObject))) return true;

    // Looks like they are not the same object, so the compare result is false
            return false;
        }
    }
```

47.2.2　对象的字符串表示

许多情况下,特别是在调试期间,我们需要把一个对象打印输出到一个输出流。Object 中的 toString()方法正是为这一目的而建立的,其格式为:

```
public String toString()
```

toString()的缺省实现打印出对象的类名和其散列码。在自己的对象中我们可能还需要提供其它信息。例如,如果我们定义了一个 Employee 对象,或许需要 toString()方法打印出雇员的 ID 号:

```
public String toString()
{
    return "Employee #" + this.employeeID;
}
```

toString()方法对于创建对象的字符串表示很方便。该方法并不是打算用来保存一个对象中所有信息的一种机制,所以也就不存在对应方法 fromString()。

47.2.3　克隆对象

clone()方法创建对象的一个复制拷贝。被克隆的对象必须支持 Cloneable 接口。Cloneable 接口本身并不包含任何方法——它只是作为一个指示器用来表示一个对象可以进行克隆。通过在 clone 方法抛出一个 CloneNotSupportedException 异常,一个对象可以选择实现 Cloneable 接口,但仍不支持克隆操作。clone() 方法的格式为:

```
protected Object clone()
    throws CloneNotSupportedException,OutOfMemoryError
```

由于 clone()方法仅拷贝原始数据类型和对象的引用,所以有时候我们需要创建自己的 clone()方法。例如,对下面的类:

```
public class StoogesFilm extends Object implements Cloneable
{
    public String[ ] stooges;
    public StoogesFilm()
    {
        stooges = new String[3];
        stooges[0] = "Moe";
        stooges[1] = "Larry";
        stooges[2] = "Curly";
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }
}
```

缺省的 clone()方法对 StoogesFilm 仅拷贝对 stooges 数组的引用。不幸的是如果新克隆的对象决定第三个角色(stooge)使用 Shemp 而不是 Curly，这样就要修改 stooges 数组，这将需要对两个拷贝都做修改：

```
StoogesFilm film1 = new StoogesFilm();//创建一个 StoogesFilm
System.out.println("The third stooge in film 1 is" + film1.stooges[2]);

StoogesFilm film2 =(StoogesFilm)film1.clone();
// 创建第一个影片(对象)的一个拷贝
film2.stooges[2] = "Shemp"; //用 Shemp 替换 Curly

System.out.println("The third stooge in film1 is now" + film1.stooges[2]);
System.out.println("The third stooge in film2 is" + film2.stooges[2]);
```

上述代码段的输出将会是：

```
The third stooge in film 1 is Curly
The third stooge in film 1 is now Shemp
The third stooge in film 2 is Shemp
```

我们可以通过创建一个克隆 stooges 数组的 clone()方法来解决这一问题：

```
public Object clone() throws CloneNotSupportedException
{
//使用缺省的克隆方法创建一个初始的克隆对象
    StoogesFilm returnValue = (StoogesFilm)super.clone();
//创建 stooges 数组的一个单独的拷贝
    returnValue.stooges = (String[ ])stooges.clone();
    return returnValue;
}
```

当增加了这一方法后，上述代码段的输出变为：

```
The third stooge in film 1 is Curly
The third stooge in film 1 is now Curly
The third stooge in film 2 is Shemp
```

47.2.4 析构化

当一个对象即将被垃圾收集器从内存中删除时，调用 finalize()方法。一般情况下，我们的对象不必重载 finalize()方法，但是如果我们分配了 Java 虚拟机之外的资源(通常使用本地方法分配)，则需要实现一个 finalize()方法，以释放这些资源。finalize()方法的格式为：

```
protected void finalize() throws Throwable
```

警告：必须在我们自己的 finalize()方法中某处调用 super.finalize，否则，由父类所分配的资源将不能被正确释放。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

典型的 finalize()方法如下所示：

```
protected void finalize() throws Throwable
{
    super.finalize();  //在一个析构化方法中必须有这一调用
//(释放外部资源的其它代码)
}
```

47.2.5　系列化对象

系列化对象这一概念出现于 Java API 1.1 版中，对象系列化是关于存放在对象中的数据的存储及提取的。我们可以使用对象系列化将一个对象的内容保存在一个文件中或在网上发送。我们可以保护属性不被系列化。例如，我们或许有一个打开文件的句柄，当在其它一些系统中提取对象时，该句柄可能会不起作用。我们可以将一些属性标记为临时的(transient)，以防止系统系列化这些属性。例如，假设有一个 InputStream，我们不希望对它进行系列化，则可以将其声明为 transient：

```
public transient InputStream myStream;
//不要系列化该属性
```

对于需要进行系列化的对象，必须实现 java. io. Serializable 接口。与 Cloneable 接口一样，java.io.Serializable 接口中也不含有任何方法。它只是作为一个标记说明一个对象可以被系列化。奇怪的是，虽然 java.io.Serializable 接口中不含有任何方法，但是如果我们的对象需要自定义系列化时，必须实现两个方法：

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException
```

readObject()和 writeObject()方法必须完全按照上述方式声明。系列化代码包含对这些方法的特殊检查。我们可能不必实现自己的 readObject()和 writeObject()方法，但是最好能够知道，如果需要的话还是可以的。有些情况下，系列化一个对象时可以走一些捷径，或者可能需要防止对象中的某些属性不被系列化，但又不希望将这些属性标记为 transient(例如，我们可能希望这些属性的系列化独立于对象的当前状态)。

47.2.6　散列码

对象的 hashCode()方法返回一个对每个对象来说是唯一的整数。hashCode()的值由 Hashtable(散列表)在存储和提取对象时使用。通常我们可能只依靠缺省的实现，但是也有这样的情况，即我们有一个更好的方法计算对象的散列码。hashCode()方法格式为：

```
public int hashCode()
```

散列表是一个使用非数值"键字"作为其索引的相关数组。换句话说，它类似于索引值不是数值而是其它一些信息的数组。散列表使用散列码将对象分组为桶(bucket)。当搜索一个对象时，只要使用桶搜索对象的散列码即可。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

警告:如果我们创建自己的 hashCode()方法,必须确保对两个相等的对象返回相同的散列值。Hashtable 类使用 hashCode()方法,以帮助定位相同的对象,如果两个对象的散列值不同,Hashtable 类将认为这两个对象是不同的,并且甚至将永远不再检查 equals()方法。

47.2.7 wait()和 notify()

wait()和 notify()方法为在相互独立的线程中运行的对象相互通知某些事件提供了一种通信手段。例如,一个对象可能正在向一个数组中写入信息,而另一个对象可能正在从该数组中读取数据。读取数据的对象调用 wait()方法,等待到写入数据的对象完成写入。当写入完成时,写入者调用 notify()方法,通知读取者它已经写入完毕。为有效起见,读取者线程应休眠直到它从写入者线程那里接收到通知信号。这种思想看似简单,但有几点应特别注意:

- 如果在一个对象开始等待前调用了 notify()方法,那么这种通知将被忽略。这可能会引起一个对象无休止地等待一个不可能到来的信号。考虑下面的情景。一个叫做 Bob 的对象准备等待天空变蓝,对象 Sun(太阳)通知 Bob 天空是蓝的(就像一贯的做法一样)。这将是正常情况,一切运转正常,但当发生下面的情况时出现例外。

程序启动,Sun 自动通知 Bob 天空是蓝的,但此时 Bob 尚未开始等待,所以通知被忽略。片刻之后 Bob 进入其等待状态。不幸的是,Sun 已经发出过通知,这样,Bob 将永远地等待下去。为了解决这一问题,我们应该设置一个标志,用来指示一个对象是否应该等待。如果当 Sun 第一次通知 Bob 时,一个变量 skyIsBlue 被设置为 true(真),Bob 在启动 wait()之前检查该变量,这样它就不用再启动等待,问题得以解决。

- notify()和 wait()必须在同步方法或块中调用。另外,这些方法不能被运行它的线程以外的其它线程所调用。仍以上面的 Sun 和 Bob 为例,Sun 不能使用 Bob.wait()命令,如果使用了,将会产生一个 IllegalMonitorStateException 异常。
- 由于可以使用异常来中断 wait(),所以我们应该将 wait()放在一个 while 循环中,检查等待标志然后再调用 wait()。

wait()方法有三种形式:

```
public final void wait()
    throws InterruptedException, IllegalMonitorStateException
```

该方法将永远等待,直到 notify()发出信号。

```
public final void wait(long timeout)
    throws InterruptedException, IllegalMonitorStateException
```

该方法等待 notify()发出信号,到 timeout 秒时限后返回。

```
public final void wait(long timeout,int nano)
    throws InterruptedException, IllegalMonitorStateException
```

该方法等待 notify()发出信号,到 timeout 秒和 nano 毫秒时限后返回。

notify()方法有两种形式:

```
public final void notify() throws IllegalMonitorStateException
```

该方法向等待这一对象的线程发送一个通知,如果有多个线程等待,它将向等待时间最长

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

的线程发送通知。

public final void notifyAll() throws IllegalMonitorStateException

该方法向所有等待的线程发送通知。

警告：notify()、notifyAll()及 wait()方法必须从同步方法或同步块中调用。另外，notify()必须从同步于相应的 wait()的同一个对象的方法或块中调用。换句话说，如果某个对象 myObject 调用 wait()，另一个对象调用 myObject.notify()，那么调用块或方法必须与 myObject 同步。见如下的伪代码：

```
public class MyObject{
    public synchronized wait(){
        wait()
    }
}
public class AnotherObject{
    public void notifyIt(MyObject myObject){
        synchronized(myObject){
            myObject.notify();
        }
    }
}
```

程序清单 47.2 给出了一个使用 wait()和 notify()实现一个信号系统的实例。

```
/**
 * This class provides a signaling mechanism for objects.
 * An object wishing to send a signal calls the signal method.
 * An object receiving the signal would wait for a signal with
 * waitForSignal.  If there is no signal pending, waitForSignal
 * will wait for one.  If there are multiple signals sent, the
 * class will keep track of how many were sent and will not call
 * wait until there are no more pending signals.
 * There should only be one object waiting for a signal at any given
 * time.

 */

public class Signaler extends Object
{
    protected int signalCount;      // the number of pending signals
    protected boolean isWaiting;    // is an object waiting right now?
    protected boolean sentNotify;   // Did someone send a notify?
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
/**
 * Creates an instance of a signaler
 */
    public Signaler()
    {
        signalCount = 0;      // no pending signals
        isWaiting = false;    // no one waiting
    }

/**
 * Sends a signal to the object waiting for a signal.
 * @exception Exception if there is an error sending a notification
 */
    public synchronized void signal()
    throws Exception
    {
        signalCount++;        // Increment the number of pending signals
        if (isWaiting)        // If an object is waiting, notify it
        {
            try {
                sentNotify = true;
                notify();
            } catch (Exception IllegalMonitorStateException) {
                throw new Exception("Error sending notification");
            }
        }
    }

/**
 * Waits for a signal.  If there are signals pending, this method will
 * return immediately.
 */
    public synchronized void waitForSignal()
    {
        while (signalCount == 0)      // If there are no signals
                                      // pending, wait for a signal
        {
            sentNotify = false;
            isWaiting = true;         // Yes, someone is waiting

// Want to keep looping until a notify is actually sent, it is possible
// for wait to return without a notify, so use sentNotify to see if we
// should go back to waiting again.
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            while (! sentNotify)
            {
                try {
                    wait();
                } catch (Exception waitError) {
                    // Shouldn't really ignore this, but...
                }
            }
            isWaiting = false;      // I'm not waiting any more
        }
        signalCoun - - ;      // one fewer signal pending
    }
}
```

如果熟悉 Java 的 synchronized 方法修饰符，可能会对 Signaler 类调用 notify()感到奇怪。如果并不感到奇怪，那么可能是知道其原因，也可能是没有看出问题。问题是 waitForSignal()和 signal 都是同步的。如果某个线程在 waitForSignal()方法中间调用 wait()时被阻塞，由于同步锁定的原因，Signal()方法不能被调用。Signaler 类能够工作的原因是当调用 wait()方法后，wait()释放同步锁定，当返回后，又重新获得锁定。

47.2.8　获取对象的类

使用 getClass()方法，我们可以检索对应于对象的那个类的 Class 对象的实例：

```
public final Class getClass()
```

47.3　Class 类

Class 类包含有描述 Java 类的信息。Java 中的每一个类都有一个相应的 Class 实例。甚至也有一个描述 Class 类本身的 Class 实例，如果试用下述的代码行，我们会不知道将发生什么样的情况：

```
Class newClass = new Class();
```

实际上这是不允许的，对 Class 对象不存在公共构造器。然而，我们可以使用下述三种方式获取 Class 的一个实例：

- 在一个对象中使用 getClass()方法获取该对象的 Class 实例。
- 使用 Class 中的静态方法 forName()，用类的名字获得一个 Class 实例。
- 使用一个定制的 ClassLoader 对象加载一个新类。

47.3.1　动态加载

Class 类是一个强有力的结构，它允许我们实现一些在 C ++ 中无法完成的工作。

一般地，我们使用如下的语句实例化一个类：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Object vehicle = new Car();
```

然而,假设我们喜欢使用需实例化的类的名字创建一个 vehicle,我们可能会按如下的方式进行:

```
String vehicleClass = (代表一个类名的某字符串)
    Objet vehicle;
    if(vehicleClass.equals("Car")
    {
        vehicle = new Car;
    }
    else if (vehicleClass.equals("Airplane"))
    {
        vehicle = new.Airplane();
    }
```

这样是可行的,但仍不够灵活。假设我们增加一个新类 Train,我们不希望再增加一个 else if 语句来检查 Train()。这正是引入 Class 的原因。使用 Class.forName()和 Class.newInstance(),我们可以执行与上述代码相同的工作:

```
Object vehicle;
//首先使用 vehicleClass 获得类名字
Class whichClass = Class.forName(vehicleClass);
//现在请求类创建一个新实例
vehicle = whichClass.newInstance();
```

Class 中的 forName()方法定义如下:

```
public static Class forName(String className)
    throws ClassNotFoundException
```

它返回对应于 className 的 Class 实例,或产生一个 ClassNotFoundException 异常。newInstance()方法定义如下:

```
public Object newInstance()
    throws InstantiationException,IllegalAccessException
```

它返回类的一个新实例,或实例化类时发生错误产生一个异常。

警告:也可以只使用 newInstance()实例化提供有空构造器(不带参数的构造器)的对象。如果试图使用 newInstance()实例化一个没有空构造器的对象,则将获得一个 NoSuchMethodError 错误。我们应当对 NoSuchMethodError 进行捕捉,但请记住这是一个错误而不是异常,只捕捉 Exception(异常)是抓不住它的。

47.3.2 获取关于类的信息

我们也可以使用 Class 获取关于一个类的信息:

```
public String getName()
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

该方法返回类的名字。

```
pulbic boolean isInterface()
```

如果类实际上是一个接口，该方法返回 true(真)。

```
public Class getSuperclass()
```

该方法返回类的父类。

```
public Class[] getInterface()
```

该方法返回一个数组，其中包含的是类所支持的每个接口的 ClassLoader 的实例。

Java 1.1 的 Reflection API 向 Class 类添加了一些方法，用于检测一个类的属性和方法，这些方法是：

```
public Class getInterfaces()
public Class getComponentType()
public int getModifiers()
public Class getDeclaringClass()
public Class[] getClasses()
public Field[] getFields() throws SecurityException
public Method[] getMethods() throws SecurityException
public Constructor[] getConstructors() throws SecurityException
public Field getField(String name)
    throws NoSuchFieldException, SecurityException
public Method getMethod(String name,Class parameterTypes[])
    throws NoSuchMethodException,SecurityException
public Constructor getConstructor(Class parameterTypes[])
    throws NoSuchMethodException,SecurityException
public Class[] getDeclaredClasses() throws SecurityException
public Field[] getDeclaredFields() throws SecurityException
public Method[] getDeclaredMethods() throws SecurityException
public Constructor[] getDeclaredConstructors() throws SecurityException
public Field getDeclaredField(String name)
    throws IllegalArgumentException,SecurityException
public Method getDeclaredMethod(String name,Class parameterTypes[])
    throws NoSuchMethodException,SecurityException
public Constructor getDeclaredConstructor(Class parameterTypes[])
    throws NoSuchMethodException,SecurityException
```

这些 Reflection API 方法将在第 48 章“反射”中深入讨论。

47.4　Package 类

Java 1.2 API 中有一个新类叫做 Package。Package 类主要用于帮助我们确定加载到系统中的每个软件包版本，一般地，版本信息实际存放在特定软件包的说明(Manifest)文件中。

有两种方法可获得一个 Package。getPackage()方法只获得一个指定的软件包，而 getAll-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Package()返回类加载器当前所认知的所有软件包的一个数组：

```
static Package getPackage(String packageName)
static Package[ ] getAllPackage()
```

一旦有了 Package 对象，我们就可以获得关于对应软件包的信息，如软件包的标题，生产软件包的厂商名称，软件包的名字(使用 getPackage() 方法可以获取该信息，而使用 getAllPackages()方法无法获取该信息)，封装软件包的源 URL，软件包所实现的规范名称，软件包的散列码(来自它的名字)，以及软件包的版本号等。

```
String getImplementationTitle()
String getImplementationVender()
String getImplementationVersion()
String getName()
URL getSealBase()
String getSpecificationTitle()
boolean isCompatibleWith(String Desired)
boolean isSealed()
String toString()
```

47.5 String 类

String 类是 Java API 中最有用的类之一，它可以使我们创建和操作字符串。必须记住，Java 字符串是永远不改变的，也就是说，我们不能修改一个字符串的内容。从表面上看，这使得 String 类看上去毫无用处。但实际上，创建可以改变的字符串又有什么好处呢？在 Java 中我们操作字符串的方法是基于其它字符串创建新的字符串。换句话说，我们不是将"father"修改成"grandfather"，而是创建了一个新的字符串"grand" + "father"。StringBuffer 类提供了直接操作字符串数据的手段。

47.5.1 创建字符串

Java 提供了许多字符串构造器：

```
public String()
```

该方法创建一个空的字符串。

```
public String(String value)
```

该方法使用指定 value 的一个拷贝创建一个新字符串。

```
public String(char[ ] value)
```

该方法使用指定值 value 中的字符创建一个新字符串。

```
public String(char[ ] value,int from,int count)
    throws StringIndexOutOfBoundsException
```

该方法使用指定值 value 从偏移值 from 开始，共 count 个字符创建一个新字符串。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public String(byte[] value,int hibyte)
```

该方法使用 value 中的字符，并以 hibyte 为每个字符的高八位值创建一个新字符串（切记，Java 字符是 16 位的，而不是 C 中的 8 位）。

```
public String(byte[] value,int hibyte,int from,int count)
    throws StringIndexOutOfBoundsException
```

该方法使用 value 中从偏移量 from 开始，共 count 个字符，并且每个字符的高八位使用 hibyte 值，创建一个新字符串。

```
public String(StringBuffer buffer)
```

该方法使用 StringBuffer 的内容创建一个新字符串。

下面是使用不同的方法创建字符串“Foo”的例子：

```
String foo1 = new String("Foo");

char foochars[] = {'F','o','o'};
String foo2 = new String(foochars);

char foo2chars[] = {'B','a','r','e','F','o','o','t'};
String foo3 = new String(foo2chars,4,3); //偏移量 4,长度 3

byte foobytes[] = {70,111,111}; //Foo 的 ascii 码
String foo4 = new String(fooBytes,0) //使用 0 作为高八位值

byte foo2bytes[] = {66,97,114,101,70,111,111,116};//BareFoot 的 ascii 码
String foo5 = new String(foo2Bytes,0,4,3);//0 作为高八位,偏移量 4,长度 3

StringBuffer fooBuffer = new StringBuffer();
fooBuffer.append('F');
fooBuffer.append("oo");
String foo6 = new String(fooBuffer);
```

String 类还提供了一些静态方法，用于从其它对象创建字符串。下面的 valueOf()方法从一个原始数据类型创建了一个字符串表示：

```
public static String valueOf(boolean b);
public static String valueOf(char c);
public static String valueOf(int i);
public static String valueOf(long l);
public static String valueOf(float f);
public static String valueOf(double d);
```

有些 valueOf()方法与 String 和 Object 中的其它方法等价。例如：

```
public static String valueOf(Object ob)
```

该方法与 Object 中的 toString()方法相同。方法

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public static String valueOf(char[ ] data);
public static String copyValueOf(char[ ] data);
```

与 String 的构造器：

```
public String(char[ ] data)
```

相同。同样，下面的方法

```
public static String valueOf(char[ ] data,int from, int count)
public static String copyValueOf(char[ ] data,int from,int count)
```

与 String 的构造器：

```
public String(char[] data,int from,int count)
```

等价。

47.5.2 字符串长度

length()方法返回一个字符串的长度：

```
public int length()
```

注意，与数组中的 length 属性不同，String 类中的 length()是一个方法。访问 length 属性的唯一情况是在数组中，而在使用标准 Java 类的任何情况下，length()都是作为方法调用的。

47.5.3 比较字符串

由于字符串是 Java 对象，我们可以使用 = = 和 equals()方法比较字符串。使用 = = 比较字符串时，应特别小心。例如，在下面的代码段中：

```
String a = new String("Foo");
String b = new String("Foo");
```

而比较式 a = = b 为 false(假)，因为 a 和 b 是两个不同的对象，即使它们有相同的值。A.equals(b)比较则是 true(真)，因为它们的值都是“Foo”。

String 类还提供了一种方便且与大小写无关的比较：

```
public boolean equalsIgnoreCase(String anotherString)
```

该方法比较两个字符串，但不考虑字母的大小写，所以

"Foo".equals("FOO")为 false(假)，"Foo". equalsIgnoreCase("FOO") 为 true(真)。

如果我们想比较两个字符串按字母顺序排序后的先后次序，可使用 compareTo()方法：

```
public int compareTo(String anotherString)
```

如果字符串与指定的字符串相等，则该方法返回 0，如果字符串在指定的字符串 anotherString 之前，该方法返回一个小于 0 的数；如果字符串在指定的字符串 anotherString 之后，该方法返回一个大于 0 的数。例如，"foo".compareTo("bar")将返回一个正数，因为"foo"排在"bar"之后。

我们也可以对字符串的一部分进行比较。如果一个字符串以另一个字符串开始，则 startsWith()方法返回 true(真)：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public boolean startsWith(String anotherString)
```

startsWith()方法的一个变形是从某个位置开始进行比较，如果一个字符串与另一个字符串从某个位置开始相匹配，则该方法返回 true(真)：

```
public boolean startsWith(String anotherString,int offset)
```

例如，"barefoot".startsWith("foo",4)将返回 true(真)，因为"foo" 出现在"barefoot"从 4 开始的位置上(注意，字符串偏移量从 0 开始)。

我们也可以使用 endsWith()方法检查一个字符串是否以另一个字符串结束：

```
public boolean endsWith(String anotherString)
```

有时候我们也需要将一个字符串的一部分与另一个字符串中的一部分进行比较，此时可使用 regionMatches()方法：

```
public boolean regionMatches(int from, String anotherString,
    int otherFrom, int len)
```

该方法将字符串从偏移量 from 开始的字符，与另一个字符串 anotherString 从偏移量 otherFrom 开始的字符进行比较，比较的长度为 len 个字符。

我们也可以使用 regionMatches()的另一个版本进行大小写无关的比较：

```
public boolean regionMatches(boolean ignoreCase, int from,
    String anotherString,int otherFrom, int len)
```

这一版本的 regionMatches() 与前一版本的 regionMatches() 之间的唯一区别是 ignoreCase 参数，当该参数设置为 true(真)时，所进行的比较忽略字母的大小写，如 a 和 A 被认为是一样的。

47.5.4　搜索字符串

许多情况下我们需要查找某一给定的字符串或字符是否出现在一个字符串中，如果是，其具体位置在什么地方。indexOf()方法可在一个字符串中搜索一个字符或字符串，并返回其第一次出现的偏移量：

```
public int indexOf(int ch)
public int indexOf(String anotherString)
```

这些方法返回字符串中第一个匹配所发生的位置，或者如果没有找到字符或字符串，返回 -1。由于我们有可能需要搜索除第一次匹配以外更多的匹配，所以可以指定搜索起始位置来调用 indexOf()方法：

```
public int indexOf(int ch,int startingOffset)
public int indexOf(String anotherString,int startingOffset)
```

lastIndexOf()方法执行类似的搜索，只是它从字符串的尾部开始，反向搜索：

```
public int lastIndexOf(int ch)
public int lastIndexOf(String anotherString)
```

我们也可以给出开始搜索的偏移量，使 lastIndexOf()从指定偏移量开始反向搜索：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public int lastIndexOf(int ch,int startingOffset)
public int lastIndexOf(String anotherString,int startingOffset)
```

47.5.5 提取部分字符串

String 类提供了若干个从一个字符串中提取部分字符串的方法。charAt() 方法允许我们从字符串中提取指定偏移量 index 位置上的字符：

```
public char charAt(int index) throws StringIndexOutOfBoundsException
```

例如，"bar".charAt(1)返回字符"a"。使用 toCharArray()我们可以以字符数组形式获得一个完整的字符串：

```
public char[ ] toCharArray()
```

切记，toCharArray()所返回的数组是字符串中字符的一个拷贝。不允许通过修改数组来改变字符串的内容。substring()方法返回从 index 偏移量开始的部分字符串：

```
public String substring(int index)
```

我们也可以使用一个结束索引来调用 substring()，这一版本的 substring()返回从 startIndex 开始到 endIndex(但不包括 endIndex)的部分字符串：

```
public String substring(int startIndex,int endIndex)
    throws StringIndexOutOfBoundsExcepton
```

47.5.6 修改字符串

虽然在 Java 中确实不允许我们修改字符串，但是有几个方法允许基于老的字符串创建新的字符串。例如，concat()方法将一个字符串添加到当前字符串末尾，并返回新的合并字符串：

```
public String concat(String otherString)
```

"foo".concat("bar")方法调用返回"foobar"。

toLowerCase()和 toUpperCase()方法返回将字符串中所有字母都转换为小写或大写的一个字符串拷贝：

```
public String toLowerCase()
public String toUpperCase()
```

"FooBar".toLowerCase()返回"foobar"，而"FooBar".toUpperCase()返回"FOOBAR"。

trim()方法删去一个字符串开始和结尾处的空白字符。空白字符包括空格、制表符、换页符、换行符及回车符，即' '、'\t'、'\f'、'\n'及'\r'：

```
public String trim()
```

例如，" Hi Ceal! ".trim()将返回"Hi Ceal!"。

最后，我们可以通过调用 replace 方法以一个指定字符替换字符串中所找到的所有另一个字符：

```
public String replace(char oldChar,char newChar)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

"fooble".replace('o','e')将返回"feeble"。

47.6　StringBuffer 类

StringBuffer 类是建立字符串的一个工作台。它包含了添加新字符到缓冲区,然后将最终结果转换为一个字符串的一些方法。与 String 类不同,当我们添加字符到一个 StringBuffer 时,并没有产生 StringBuffer 的一个新拷贝。这使得字符串的构造更为有效。

47.6.1　创建 StringBuffer

创建一个 StringBuffer 最简单的方法是使用空构造器:

```
public StringBuffer()
```

我们也可以使用一个初始长度创建一个 StringBuffer:

```
public StringBuffer(int length)
```

最后,我们也可以使用一个字符串来创建一个 StringBuffer,此时该字符串的内容拷贝到了 StringBuffer 中:

```
public StringBuffer(String str)
```

47.6.2　添加字符到一个 StringBuffer 中

insert()方法允许我们向一个 StringBuffer 中插入字符、字符串及数字。我们可以使用下面的 insert()方法之一插入原始数据类型之一的字符表示:

```
public StringBuffer insert(int offset, boolean b) throws StringOutOfBoundsException
public StringBuffer insert(int offset, char c) throws StringOutOfBoundsException
public StringBuffer insert(int offset, int i) throws StringOutOfBoundsException
public StringBuffer insert(int offset, long l) throws StringOutOfBoundsException
public StringBuffer insert(int offset, float f) throws StringOutOfBoundsException
public StringBuffer insert(int offset, double d) throws StringOutOfBoundsException
```

在以上每个方法中,偏移量参数指的是 StringBuffer 中字符应该插入的位置。这些方法所返回的 StringBuffer 实例并不是原 StringBuffer 的一个拷贝,而是它的一个新的引用。我们可以放心地不去考虑它的返回值。

我们可以使用下面的方法将一个字符串插入到一个 StringBuffer 中:

```
public StringBuffer insert(int offset,String str)
    throws StringOutOfBoundsException
```

我们也可以使用下述方法插入一个对象的字符串表示:

```
public StringBuffer insert(int offset,Object ob)
    throws StringOutOfBoundsException
```

这一方法使用了 Object 中的 toString()方法创建指定对象的一个字符串表示。最后,我们可以使用下述方法将一个字符数组插入到一个 StringBuffer 中:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public StringBuffer insert(int offset,char[] data)
    throws StringOutOfBoundsException
```

对每个 insert()方法都有一个相应的 append()方法，添加字符到 StringBuffer 末尾处：

```
public StringBuffer append(boolean b)
public StringBuffer append(char c)
public StringBuffer append(int i)
public StringBuffer append(long l)
public StringBuffer append(float f)
public StringBuffer append(double d)
public StringBuffer append(String str)
public StringBuffer append(Object ob)
public StringBuffer append(char[] data)
```

47.6.3 StringBuffer 的长度

StringBuffer 有两个长度概念：

- 缓冲区当前的字符个数
- 缓冲区的最大容量

length()方法返回当前缓冲区中的字符总数：

```
public int length()
```

capacity()方法返回缓冲区的最大容量：

```
public int capacity()
```

当我们添加字符时 StringBuffer 会自动增长，那么为什么我们还要关心它的容量呢？当缓冲区增长时，它必须分配更多的内存。如果我们指定缓冲区的容量至少大到我们所希望的字符串大小，我们应避免分配附加空间开销。ensureCapacity()方法指出 StringBuffer 必须能够存储的最小字符个数：

```
public void ensureCapacity(int minimumAmount)
```

使用 setLength()方法可以修改 StringBuffer 的长度：

```
public void setLength(int newLength)
    throws StringOutOfBoundsException
```

如果所指定的新长度小于原来的长度，那么新长度以外的字符将丢失。

47.6.4 在 StringBuffer 中获取和设置字符

我们可以使用 charAt()和 setCharAt()方法，在一个 StringBuffer 中进行单个字符操作。CharAt()方法返回 StringBuffer 中特定的偏移量位置上的字符：

```
public char charAt(int offset)
    throws StringIndexOutOfBoundsException
```

setCharAt()方法修改缓冲区中特定偏移量位置上的字符：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public void setCharAt(int offset,char newchar)
    throws StringIndexOutOfBoundsException
```

getChars()方法允许我们从一个 StringBuffer 中拷贝一个范围内的字符到一个字符数组中。我们必须指定 StringBuffer 中的开始和结束偏移量,目的字符数组,以及拷贝到数组中的偏移量位置:

```
public void getChars(int beginOffset,int endOffset, char[] dest,
    int destOffset) throws StringIndexOutOfBoundsException
```

47.6.5　从 StringBuffer 创建一个字符串

当我们在 StringBuffer 中建立完一个字符串之后,可以使用 toString() 方法将其转换到一个 String 对象中,该方法覆盖了 Object 类中的 toString()方法:

```
public String toString()
```

注意:当我们使用 StringBuffer 中的 toString 方法创建字符串时,String 和 StringBuffer 共享同一缓冲区,以避免额外的拷贝。如果 StringBuffer 随后被修改,它将在进行修改之前首先对缓冲区建立一个拷贝。

47.7　Thread 类

在 Java 程序中,一个线程代表一个单一的执行线程。每个线程都有一个相关的 Runnable (可执行)接口,这个接口可以是线程本身。Runnable 接口实现了 run()方法,该方法包含线程所执行的代码。可以将线程想象为驱动 Runnable 接口的一个马达。线程还有一个概念叫做线程组,线程组实现了线程的安全策略。我们不希望有什么"流氓"线程将其它线程置于休眠,或修改其它线程的权限。一个线程应该只能操作其自己线程组中的线程,或其自己线程组的一个子组中的线程。

47.7.1　创建线程

使用空构造器可以创建一个线程:

```
public Thread()
```

当我们创建线程而不指定 Runnable 接口时,该线程使用其自身作为 Runnable 接口。Thread 类中 run()方法的缺省实现不做任何事情。为了指定其它 Runnable 接口,可使用 Thread 构造器的变型:

```
public Thread(Runnable target)
```

当线程启动后,它将激活 target 中的 run()方法。当我们创建了一个线程后,它便增加到当前线程的线程组中。如果需要使该线程属于其它的组,必须在创建线程时指出:

```
public Thread(ThreadGroup group,String name)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

name 参数是一个可选参数，它指的是一个线程名，用来使线程分离。如果我们不想使用它，可使用 null 代替线程名。其它的线程构造器是以上几个构造器的各种组合：

```
public Thread(String name)
public Thread(Runnable target,String name)
public Thread(ThreadGroup group,Runnable target)
public Thread(ThreadGroup group,Runnable target, String name)
```

47.7.2 启动和结束线程

start()和 stop()方法控制线程的初始启动和最后结束。

```
public synchronized void start()
    throws IllegalThreadStateException
public final void stop()
public final void stop(Throwable stopThrowable)
```

如果线程已经处于运行中，则 start()方法产生一个 IllegalThreadStateException 异常。在 JDK 1.2 之前，一般情况下通过调用不带参数的 stop()方法来终止一个线程，该方法向线程抛出一个 ThreadDeath 错误。使用第二个变型的 stop()方法，可以抛出除 ThreadDeath 以外的其它信息。除调用自己的 stop()方法，线程也可以抛出一个 ThreadDeath 错误。然而，在 Java 1.2 中，我们不应该再调用 stop()方法，原因是 stop()本身并不安全，因为如果线程有一些同步的对象或对某些稀有资源加锁，则有可能引起死锁。相反，我们应确保 run()方法能够终止。

警告：Java 允许我们对 ThreadDeath 错误进行捕捉，这只在极少数的情况下即 finalize()方法不能满足要求的情况下，才这样做。如果我们捕获到 ThreadDeath，必须保证将它再抛出，否则，我们的线程将不会终止。

47.7.3 等待线程完成

假设我们有一个应用程序，该应用程序要进行一些繁重的计算，在使用这些计算结果之前还需要做一些其它工作。那么，我们可以将这些繁重的计算分离出来，建立一个其自己的线程。但是我们如何知道线程结束呢？我们可以使用 join() 方法：

```
public final void join() throws InterruptedException
```

下面的代码段演示了使用 join()方法的一种情况：

```
Double finalResult; //用于计算线程存储结果
Thread computeThread = new HeavyComputationThread(finalResult);
ComputeThread.start();
//进行其它操作
ComputeThread.join();//等待繁重计算的完成
```

由于我们并不希望永远等待下去，所以 join()也支持一个时限值：

```
public final synchronized void join(long millis)
    throws InterruptedException
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public final synchronized void join(long millis,int nanos)
    throws InterruptedException
```

这些 join()方法只等待 millis 毫秒,或 millis 毫秒 + nanos 毫微秒。

47.7.4　休眠与让出

使用线程时我们需要做的最常见的事情是使线程暂时停止片刻。如果在继续进行操作之前需要线程等待一段时间,可使用 sleep()方法:

```
public static void sleep(long millis)
    throws InterruptedException
public static void sleep(long millis,int nanos)
    throws InterruptedException
```

这些方法挂起线程的执行 millis 毫秒,或 millis 毫秒 + nanos 毫微秒。sleep()方法常用于动画循环中:

```
public void run()
{
    while(true){  //动画永远播放下去
        changeCurrentFrame();  //播放下一帧动画
        repaint();//重画屏幕
        try{
            sleep(100);
        }catch(InterruptedException insomnia){
            //休眠时获得中断
        }
    }
}
```

如果我们有一个线程,在执行大量计算时过多地占用了 CPU,这时可能需要该线程让出一些 CPU 的执行时间给其它线程。对此,我们可以使用 yield()方法:

```
public static void yield()
```

例如,假设我们有一个如下所示的循环:

```
int sum = 0;
for(int i = 0; i< 10000; i++){
    for(int j=0; j< 10000; j++){
        sum = sum + (i * j);
    }
}
```

这一循环将会运行很长时间,并且如果它在一个大程序中作为一个线程运行,它将会长时间占用 CPU。如果我们在其内层循环的后面调用一个 Thread.yield(),则该线程将会礼貌地不时为其它线程让出 CPU:

```
int sum = 0;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
for(int i = 0; i<10000; i++){
    for(int j = 0; j<10000; j++){
        sum = sum+(i*j);
    }
    Thread.yield(); //给其它线程一个运行机会
}
```

注意:实际上并不需要我们来调用 yield()方法来给予其它线程运行的机会。大多数 Java 实现支持预先空出模式,允许其它线程不时地得到运行,即使是对上述那样的一个循环。然而并不是所有的实现都支持预先空出模式,所以策略地使用 yield()方法将有助于这些实现平滑地运行。

47.7.5 守护线程

Java 程序通常运行到其所有线程都终结为止。然而有时候我们会有一个运行于后台的线程,执行一些从不会结束的清理和维护任务。我们可以将这种线程标记为守护线程,以告诉 Java 虚拟机在检查所有线程是否结束时忽略这种线程。换句话说,Java 程序将运行到其所有非守护线程终结为止。非守护线程也称为用户线程。

注意:守护(daemon)一词的发音为"day - mon"或"dee - mon",它可以追塑到前 UNIX 时期,用来代表"磁盘与执行监视(Disk And Execution Monitor)"。在 UNIX 系统下,一个守护(daemon)是一个运行于后台的用来执行一些有用服务的程序,它与 Java 的守护线程概念类似。

要将一个线程标记为守护线程,使用 setDaemon 方法:

```
public final void setDaemon(boolean on)
    throws IllegalThreadStateException
```

on 参数为 true(真)时标记线程为守护线程,为 false (假)时标记线程为用户线程。我们可以在线程执行期间随时改变这种设置。如果是一个守护线程,isDaemon()方法返回 true(真),如果是用户线程该方法返回 false(假):

```
public final boolean isDaemon()
```

47.7.6 线程优先级

Java 的线程模式非常简单。当一个线程阻塞时,即线程挂起、休眠或等待其它事情时,Java 会从等待运行的线程集中找出一个新的线程来运行。Java 所找出的线程具有最高的优先级,如果有多个线程具有最高的优先级它将从这些线程中任意找出一个。设置一个线程的优先级使用 setPriority()方法:

```
public final setPriority(int newPriority)
    throws IllegalArgumentException
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

线程的优先级必须为 Thread.MIN _ PRIORITY 与 Thread.MAX _ PRIORITY 之间的数字。所有在这一范围之外的设置都将引发一个 IllegalArgumentException 异常。缺省情况下指定给线程的优先级为 Thread.NORM _ PRIORITY。使用 getPriority()方法可以查询一个线程的优先级：

```
public final int getPriority()
```

47.7.7　获得线程信息

Thread 类中有一些静态的方法，用来帮助我们检查运行于线程组中的当前线程和其它线程。CurrentThread()方法返回一个当前运行线程的 Thread 对象：

```
public static Thread currentThread()
```

dumpStack()方法打印一个当前线程的堆栈跟踪：

```
public static void dumpStack()
```

使用 countStackFrames()方法可以查找出一个线程所具有的堆栈帧个数。这正是 dumpStack()方法将要卸出的帧个数：

```
public int countStackFrames()
```

由于 countStackFrames()方法是一个 instance()方法，而 dumpStack()方法是一个用来卸载当前线程的堆栈帧的静态方法，所以下述的调用总是返回堆栈帧的个数，这些堆栈帧在一个 dumpStack()立即调用中将被卸出：

```
int numFrames = Thread.currentThread().countStackFrames();
```

enumerate()方法则使用当前线程组中的所有 Thread 对象填充一个数组：

```
public static int enumerate(Thread[] threadArray)
```

由于我们必须自己分配 threadArray，所以我们需要知道所返回的线程个数。ActiveCount()方法可以告诉我们当前线程组中活动的线程个数：

```
public static int activeCount()
```

程序清单 47.3 中的程序显示出当前线程组中的所有线程。

程序清单 47.3　DumpThreads.java 的源代码

```
public class DumpThreads
{
    public static void main(String[] args)
    {
// Find out how many threads there are right now
        int numThreads = Thread.activeCount();

// Allocate an array to hold the active threads
        Thread threadArray[] = new Thread[numThreads];
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
// Get references to all the active threads in this thread group
        numThreads = Thread.enumerate(threadArray);

// Print out the threads
        for (int i=0; i < numThreads; i++) {
            System.out.println("Found thread: " + threadArray[i]);
        }
    }
}
```

47.8 ThreadGroup 类

ThreadGroup 类实现了一种安全策略，它只允许同一个线程组中的线程相互修改。例如，一个线程可以修改与其处于同一组中的另一线程的优先级或将另一线程置于休眠。如果没有线程组，一个线程就会在 Java 运行环境下进行破坏活动，休眠其它线程，或更糟的是终止其它线程。线程组是以层次形式组织的，其中的每一个线程组都有一个父组。一个线程可以修改其自己所在的线程组中的其它线程，或其线程组的子组中的任何线程。只要给出线程组的名字就可以创建一个线程组：

```
public ThreadGroup(String groupName)
```

我们也可以创建一个现有线程组的一个子线程组：

```
public ThreadGroup(ThreadGroup existingGroup, String groupName)
    throws NullPointerException
```

ThreadGroup 的有些操作与 Thread 相同，只是这些操作是对线程组中所有线程进行操作的：

```
public final synchronized void suspend()
public final synchronized void resume()
public final synchronized void stop()
public final void setDaemon(boolean daemonFlag)
public final boolean isDaemon()
```

我们也可以对线程组中每个线程所具有的最高优先级进行限制，可以调用 setMaxPriority()方法：

```
public final synchronized void setMaxPriority(int priority)
```

使用 getMaxPriority()方法，可以查询一个线程组的最高优先级：

```
public final int getMaxPriority()
```

使用 getParent()方法，我们可以找出一个线程组的父线程组：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public final int ThreadGroup getPareant()
```

下述不同的 enumerate()方法可以使我们找出隶属于某个特定线程组的线程和线程组：

```
public int enumerate(Thread[ ] threadList)
public int enumerate(Thread[ ] threadList,boolean recurse)
public int enumerate(ThreadGroup[ ] groupList)
public int enumerate(ThreadGroup[ ] groupList,boolean recurse)
```

enumerate()方法中的 recurse 参数可使 enumerate()向下搜索所有子线程组，获得一个完整的后代列表。我们也可以使用 activeCount()和 activeGroupCount()方法，对一个线程组中活动的线程和线程组进行估算：

```
public synchronized int activeCount()
public synchronized int activeGroupCount()
```

47.9 Throwable 类

Throwable 类本身并不大，但它在 Java 中所扮演的角色非常重要。Java 中的每一个错误和异常都是 Throwable 类的一个子类。虽然我们经常要创建 Throwable 的子类，但我们可以使用下述的两个构造器之一实例化一个 Throwable：

```
public Throwable()
public Throwable(String message)
```

message 参数是一条与可抛出(throwable)相关联的可选的错误信息。我们可以使用 getMessage()方法提取一条可抛出的信息：

```
public String getMessage()
```

Throwable 类的一个方便的特性是它的 printStackTrace()方法，特别是在调试程序时该方法更有用：

```
public void printStackTrace()
public void printStackTrace(PrintStream stream)
```

这些方法打印出导致一个异常的方法调用的反向跟踪。堆栈跟踪的缺省输出流是 System.out。我们可以使用该方法的第二种形式将打印输出到所希望的任何一个流，如 System.err。也有这样的情况，即我们捕捉到一个异常，执行某些清理工作，然后我们又将该异常抛出供其它对象捕获。如果我们将所捕获的异常抛出，那么堆栈跟踪将给出异常的原始发生处。如果我们需要堆栈跟踪只显示异常首次捕获的位置——例如如果要求隐藏底层的细节——可使用 fillInStackTrace()方法：

```
public Throwable fillInStackTrace()
```

下述的代码段显示了如何使用 fillInStackTrace()方法隐藏一个异常的原始抛出位置的具体办法：

```
try{
    //进行自己所需的操作
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
}catch(Throwable somethingBadHappened){
    //进行一些清理工作
    throw somethingBadHappened. fillInStackTrace();
}
```

在这个例子中堆栈跟踪给出了异常的原始抛出位置是在 fillInStackTrace()处，而不是在 try 语句或它所调用的方法中。

47.10 System 类

System 类是一些有用的工具方法包，这些方法一般用于处理运行环境。System 类中有些方法在 Runtime 类中也可以找到。

47.10.1 系统输入输出流

System 类包含三个使用频繁的公共数据流：

```
public static InputStream in
public static PrintStream out
public static PrintStream err
```

C 程序员可以将这些流看成是 stdin、stdout 和 stderr 在 Java 中的对等物。当我们运行一个 Java 应用程序时，这些流通常在启动窗口读取或写入信息。我们最好不要在 Java 小程序中使用 System.in 流，因为不同的浏览器会以不同的方式对待这个流。同样对于 System.out 和 System.err 流，Netscape 将其发送到 Java 控制台窗口，而 Appletviewer 则将其发送到 Appletviewer 启动窗口。System.err 一般情况下用来打印错误信息，System.out 则用于输出其它信息。这只是开发人员的使用习惯问题。如果愿意的话，也可以使用 System.out 打印错误信息，使用 System.err 打印其它信息。

arraycopy()方法也是 System 类中经常使用的一个成员：

```
public static void arraycopy(Object source,int sourcePosition,
    Object dest,int destPosition,int length)
    throws ArrayIndexOutOfBoundsException, ArrayStoreException
```

该方法为把一个数组的信息拷贝到另一个数组中提供了一种快速拷贝方式。它将数组 source 中从偏移量 sourcePosition 开始的 length 个元素拷贝到数组 dest 从偏移量 destPosition 开始的位置。这个方法要比使用一个循环进行单个元素的拷贝节省时间。例如让我们来看如下循环：

```
int fromArray[ ] = {1,2,3,4,5};
int toArray[ ] = new int[5];
for(int i=0; i<fromArray.length; i++){
    toArray[i] = fromArray[i];
}
```

该操作可以通过使用 arraycopy 方法以如下方式更有效地实现：

```
int fromArray[ ] = {1,2,3,4,5};
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
int toArray[ ] = new int[5];
System.arraycopy(fromArray,0,toArray,0,fromArray.length);
```

47.10.2　获取当前时间

如果我们不知道自从格林尼治时间 1970 年 1 月 1 日 0 点以来，到底溜走了多少毫秒的时间，currentTimeMillis()方法可以很轻松地告诉我们：

```
public static long currentTimeMillis()
```

除非是宝瓶座时代(the Age of Aquarius)的黎明已经出现，否则这一特殊的时间并没有任何意义。它只是原始的 UNIX 向导所选择的一个参考点。currentTimeMillis()方法最常见的用途是用来确定所花费的时间。例如，如果要想估计一下执行某个循环所占用的时间，可在循环开始执行之前获取一个时间，然后再在循环执行完之后获取一个时间，再计算这两个时间的差值即可：

```
long startTime = System.currentTimeMillis();//记录开始时间
int sum = 0;
for(int i = 0; i < 100000; i++){
    sum += i;
}
long endTime = System.currentTimeMillis();//记录结束时间
System.out.println("The loop took" +
    (endTime - startTime) + "milliseconds.");
```

注意：虽然使用 currentTimeMillis()方法可以计算出当前的日期和时间，但是获取当前的日期和时间最好使用 java.util 中的 Date 类。

47.10.3　退出虚拟机

Java 程序通常在其所有用户线程结束运行后，退出执行。如果我们启动了大量的线程后决定退出程序，那么我们无须一一地去结束这些线程，只要调用 System.exit 使虚拟机(VM)结束即可：

```
public static void exit(int exitCode)
```

exitCode 参数是 VM 结束时所使用的退出码。该方法只能在 Java 应用程序中使用，Java 小程序一般禁止调用该方法，如果在小程序中使用这一方法将抛出一个 SecurityException 异常。

47.10.4　获得系统属性

系统属性基本上可以认为是 Java 环境变量的对等物。当启动一个 Java 应用程序时，我们可以对 Java 应用程序读取的属性进行定义。getProperty()方法返回相应属性的字符串，如果该属性未设置则返回 null：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public static String getProperty(String propertyName)
```

为了避免每次检查 null 的麻烦，我们可以在调用 getProperty()时使用一个缺省值参数，指定在属性值未设置的情况下代替 null 的返回值：

```
public static String getProperty(String propertyName,
    String defaultValue)
```

程序清单 47.4 中的程序说明了使用 getProperty()的方式。

程序清单 47.4 PrintProperty.java 的源代码

```
public class PrintProperty extends Object
{
    public static void main(String[] args)
    {
        String prop = System.getProperty("MyProperty","My Default Value");
        System.out.println("MyProperty is set to: " + prop);
    }
}
```

当使用 java 命令运行程序时，使用 -D 选项设置属性。例如：

```
java -DMyProperty="Hi There" PrintProperty
```

该命令使应用程序打印出：

```
MyProperty is set to: Hi There
```

如果运行程序时未设置 MyProperty，则打印出：

```
MyProperty is set to: My DefaultValue
```

使用 Properties 类，getProperties()和 setProperties()方法可使我们查询和设置系统属性：

```
public static Properties getProperties()
public static void setProperties(Properties prop)
```

参见 46.5 节“Properties 类”。

47.10.5 强制垃圾收集

垃圾收集器一般情况下运行于后台，并不时地收集已不使用了的内存。使用 gc 方法可强制垃圾收集器启动：

```
public static void gc()
```

同样，我们也可以使用 runFinalization()方法强制 finalize() 方法对一个已准备被收集的对象运行：

```
public static void runFinalization()
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

47.10.6　加载动态库

当使用了一个包含本地方法调用的类时，需要装入包含该方法的库。loadLibrary()方法在库路径中搜索与指定的库名 libname 相匹配的库：

```
public static void loadLibrary(String libname)
    throws UnsatisfiedLinkError
```

如果我们已经知道库文件的完整路径，可以使用 load()方法节省一些时间：

```
public static void load(String filename)throws UnsatisfiedLinkError
```

47.11　Runtime 类和 Process 类

Runtime 类提供了很多与 System 类相同的方法，但也增加了查询可使用内存和运行外部程序的功能。与 System 类相同的 Runtime 方法有：

```
public void exit(int exitCode)
public void gc()
public void runFinalization()
public synchronized void load(String filename)
    throws UnsatisfiedLindError
public synchronized void loadLibrary(String libname)
    throws UnsatisfiedLinkError
```

> **注意**：与 System 类不同，Runtime 类的方法是非静态的，这就是说我们必须有一个调用这些方法的 Runtime 实例。创建实例时不是使用 new，而是使用 Runtime.getRuntime()方法。

47.11.1　查询可用内存

freeMemory()和 totalMemory()方法分别用来获得我们可使用的自由内存和 Java 虚拟机(VM)可使用的内存：

```
public long freeMemory()
public long totalMemory()
```

下述的代码段打印出自由内存的百分比：

```
Runtime r = Runtime.getRuntime();
int freePercent = 100 * r.freeMemory()/r.totalMemory();
System.out.println(freePercent + "% of the VM's memory is free.");
```

47.11.2　运行外部程序

尽管 Java 是一个功能强大的编程环境，但偶尔也会出现必须从我们的应用程序中运行

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

个外部程序的情况。exec()方法可以完成这一任务：

```
public Process exec(String command) throws IOException
public Process exec(String command,String[] envp) throws IOException
public Process exec(String[] cmdArray) throws IOException
public Process exec(String[] cmdArray,String[] envp)throws IOException
```

exec()方法中的 envp 参数包含了设置程序运行的环境变量。envp 中的字符串形式为 name = value。exec()所返回的 Process 实例允许我们与外部程序通信，等待外部程序执行完后，获得其退出码。下面的 Process 类中的方法返回输入输出流，用来向外部程序发送数据，或从外部程序接收数据：

```
public abstract InputStream getInputStream()
public abstract InputStream getErrorStream()
public abstract OutputStream getOutputStream()
```

getInputStream()方法返回一个连接到外部程序输出口上的流。如果外部程序是一个 Java 程序，则该输入流将接收写入到外部程序的 System.out 流中的所有信息。同样，getErrorStream()方法返回一个连接到外部程序错误输出口上的流，对于 Java 程序，它将连接到 System.err 流。getOutputStream()方法返回一个为外部程序提供输入的流。所有写入到这个流中的信息都将进入外部程序的输入流中，类似于 System.in 流。

如果想在外部程序执行结束之前杀死外部程序，可调用 destroy()方法：

```
public abstract void destroy()
```

如果想让程序自己完成，可使用 waitFor()方法，等待外部程序执行完：

```
public abstract int waitFor()throws InterruptException
```

waitFor()的返回值是外部程序的退出码。我们也可以使用 exitValue()方法检查这种退出码：

```
public abstract int exitValue()throws IllegalThreadException
```

47.12 Math 类

如果 Math(数学)类这一术语使我们感到头痛的话，那么我们最好坐下来好好看看这一节。Math 类中有许多很有用的数值常量和函数，包括简单的最小值和最大值函数以及对数和三角函数等。

47.12.1 min 和 max

min 和 max 函数分别返回两个数中较小的一个和较大的一个，它们分别有四种形式——int、long、float 及 double：

```
public static int min(int a,int b)
public static long min(long a,long b)
public static float min(float a,float b)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public static double min(double a,double b)
public static int max(int a,int b)
public static long max(long a,long b)
public static float max(float a,float b)
public static double max(double a,double b)
```

47.12.2　绝对值

绝对值函数 abs 将负数转换成正数，并保持正数不变，也有四种形式：

```
public static int abs(int a)
public static long abs(long a)
public static float abs(float a)
public static double abs(double a)
```

47.12.3　随机数

如果没有随机数发生器，要编写一个好游戏是很困难的，所以 Math 类很好地支持了随机数方法 random()：

```
public static synchronized double random()
```

random()方法返回一个 0.0 到 1.0 之间的数，还有其它的一些变型如下所示：

```
int num = (int)(10.0 * Math.random());  //随机数从 0 到 9
int num = (int)(10.0 * Math.random()) + 1;  //随机数从 1 到 10
```

参见 46.10 节“Random 类”。

47.12.4　舍入

舍入听起来是一个简单的过程，但在舍入时却有相当一些选择。首先，我们可以使用舍入将一个浮点数变成一个整数：

```
public static int round(float a)
```

该代码将浮点数舍入为离它最近的一个整数，也就是说 5.4 将舍入为 5，而 5.5 将舍入为 6。我们也可以将一个 double 型的数舍入为一个 long 型的整数：

```
public static long round(double a)
```

其它一些舍入方法专门用于处理 double 类型。floor()方法总是向下舍入，例如，Math.floor(4.99)返回 4.0：

```
public static double floor(double a)
```

相反，ceil 则总是向上舍入，例如 Math.ceil(4.01)返回 5.0：

```
public static double ceil(double a)
```

最后，rint()方法舍入到离该数最近的一个整数：

```
public static double rint(double a)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

47.12.5 幂和对数

与幂有关的最熟悉的函数之一是平方根，平方根由 sqrt 方法返回：

```
public static double sqrt(double a)throws ArithmeticException
```

当然，如果试图求一个负数的平方根，将会获得一个 ArithmeticException 异常。这在数学上是无意义的。

pow()方法是 x 的 y 次幂：

```
public static double pow(double x,double y)throws ArithmeticException
```

使用 pow()函数时要特别小心。如果 x = =0.0，则 y 必须大于 0。如果 x < 0.0，y 必须是一个整数。如果违反这两个条件中的任何一个，都会产生一个 ArithmeticException 异常。

提示：我们可以使用 pow()方法求一个数的 N 次根。这只要使用 pow(x,1.0/N)即可，这里的 N 是我们要求的 N 次根的 N。例如，要求平方根，使用 N = 2，这样 pow(x,1.0/2.0)返回 x 的平方根。对于立方根，N = 3，即 pow(x,1.0/3.0)。但必须记住，使用这种技术时，x 必须为一个正数。

log()方法返回一个数的自然对数：

```
public static double log(double a)throws ArithmeticException
```

作为回忆，如果 x 的自然对数等于 y，那么常量 e(大约为 2.718)的 y 次幂等于 x。例如，e 的自然对数等于 1.0，因为 e 的一次幂等于 e。1.0 的自然对数等于 0.0，因为 e 的零次幂等于 1(任何其它数的零次幂也是 1)。对于 0 和负数不能求对数，因为，没有一个幂次能使 e 等于 0，对于负数也一样。虽然我们可以使用 pow()方法求出任意一个数的任意次幂，但 Math 类为 e 的幂函数提供了一个快捷方法 exp()：

```
public static double exp(double a)
```

log()和 exp()互为反函数；它们相互抵消。即对任意 x，log(exp(x)) = = x 。同样对任意 x > 0，exp(log(x)) = = x(切记，我们不能对小于等于 0 的数使用 log())。

提示：底数为 10 的对数是另一种常用的对数类型。一个数的 log()是 e 的幂次，而以 10 为底的对数是 10 的幂次。Math 类没有提供以 10 为底的对数函数，但是我们可以使用一个简单的数学性质计算以 10 为底的对数，该性质是："一个数以 N 为底的对数是这个数的自然对数除以 N 的自然对数"。所以以 10 为底的 x 的对数为 log(x) /log(10)。如果我们需要计算以 2 为底 x 的对数，这是另一种常用的对数，也就是计算 log(x)/log(2)。

47.12.6 三角函数

人们喜爱的古老的三角函数即正弦、余弦及正切在 Math 类中也可以使用：

```
public static double sin(double angle)
public static double cos(double angle)
public static double tan(double angle)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这些函数的角度值使用弧度，它是 0 到 6.2831(2 * pi)之间的数。我们可以通过乘以 pi/180 或 0.017453 将一个角度值转换为弧度值。三角函数的角度是 6.2831 的“周期”，即某一个角度 x 与 x + 6.2831 或 x - 6.2831 是相同的，一般地，它与 x + 6.2831 * N 是相同的，N 为任意整数。

正弦函数、余弦函数及正切函数的反函数为反正弦函数、反余弦函数及反正切函数，这些方法如下：

```
public static double asin(double x)
public static double acos(double x)
public static double atan(double x)
public static double atan2(double y,double x)
```

asin()和 acos()函数返回 - 3.1415 到 3.1415 之间的一个弧度值。如果我们习惯于 0 到 6.2831 之间的弧度值，可将任何负弧度值加上一个 6.2831，这对三角函数来说是无关紧要的。atan()的准确性要稍差一些，它只返回 - 1.5708 到 1.5708 (- pi/2 到 pi/2)之间的数。而 atan2()则返回一个 - 3.1415 到 3.1415 之间的数。atan()使用一个比值 y/x 作为参数进行转换，而 atan2()则使用独立的 y 和 x 值作为参数进行转换，这使得该函数可以做额外的计算，返回完整范围 - pi 到 pi 之间的值。

47.12.7　数学常量

由于 PI 和 E 使用非常频繁，Math 类将它们定义为常量：

```
public static final double E;
public static final double PI;
```

47.13　对象包装类

大多数 Java 类更习惯于使用对象，而不是原始的数据类型。包装类提供了原始数据类型的对象版本。除了使原始数据类型看上去更像对象，这些包装类还提供了转换字符串为各种数据类型的方法及指定类型的方法。每一种包装类都包含一个静态属性称为 TYPE，它包含了包装类的 Class 对象。各种包装类中 TYPE 属性的定义都是相同的：

```
public final static Class TYPE;
```

47.14　Character 类

Character 类提供了对 char 数据类型的包装。另外，它还提供了转换字符到数值数字及转换数值数字到字符的方法。该类中还包含测试一个字符是字母、数字等的方法。Character 类的构造器只有一个，它使用表示字符的值作为其唯一参数：

```
public Character(char value)
```

使用 charValue 方法可获得存放在一个 Character 对象中的 char 值：

```
public char charValue()
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Character 类中包含有许多静态的方法,可以对字符进行分类:

方法	描述
isDigit()	一个 0~9 之间的数字
isLetter()	一个字母字符
isLetterOrDigit()	字母字符或数字
isLowerCase()	小写字母字符
isUpperCase()	大写字母字符
isJavaLetter()	字母、'$'或'_'
isJavaLetterOrDigit()	字母、数字、'$'或'_'
isSpace()	空格、新行、回车、制表符或换页符
isTitleCase()	特殊的双字母大写和小写字母

这些分类方法都返回一个 boolean 类型的值,如果字符属于相应的分类,则返回 true(真)。例如,isLetter('a')返回 true(真),但 isDigit('a')返回 false(假)。

toUpperCase()和 toLowerCase()方法返回字符的大写和小写:

```
public static char toUpperCase(char ch)
public static char toLowerCase(char ch)
```

Character 类还提供了一些数字转换方法,用来将数字转换成字符串,将字符串转换成数字:

```
public static int digit(char ch,int radix)
public static char forDigit(int digit,int radix)
```

digit()方法以指定基返回一个字符表示的数字值,(基指的是数字基,例如 10 是十进制数的基,8 是八进制的基)。例如,Character.digit('f',16)将返回 15。允许指定 Character.MIN_RADIX 到 Character.MAX_RADIX,即 2 到 36 之间的任何一个数作为基。如果对于指定的基,字符不表示一个值,则 digit()返回 -1。

forDigit() 方法以指定基转换数字值为所表示的字符。例如,Character.forDigit(6,8)将返回'6',而 Character.forDigit(12,16)将返回'c'。

47.15 Boolean 类

Boolean 类是对 boolean 数据类型的对象包装。它有两个构造器,一个以 boolean 值为参数,另一个以字符串表示的 boolean 值为参数:

```
public Boolean(boolean value)
public Boolean(String str)
```

在第二个构造器中,如果字符串的值等于 true(真)则所创建的 Boolean 类的值为 true(真),否则所创建的 Boolean 类的值为 false(假)。在比较之前字符串被转换为小写,所以 true(真)值将设置 Boolean 对象为 true (真)。使用 booleanValue()方法可以提取存储在 Boolean 对象中的 boolean 值:

```
public boolean booleanValue()
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

Boolean 类中甚至有一个 true(真)和 false(假)的对象包装版本：

```
public final static Boolean TRUE
public final static Boolean FALSE
```

valueOf( ) 方法是从一个字符串创建一个 Boolean 对象的另外一种方法：

```
public static Boolean valueOf(String str)
```

该方法与使用字符串作为参数的 Boolean 构造器是等价的。我们可以使用 getBoolean( ) 方法提取 boolean 系统参数：

```
public static boolean getBoolean(String propName)
```

该方法在系统属性中查找以 propName 指定的属性名，如果找到，将试着使用 valueOf 方法将其转换为一个 boolean 值。如果该属性的值为真，则该方法将返回 true(真)，如果该属性值为假，或不存在该属性，该方法返回 false(假)。

47.16　Number 类

对于 int、long、float 及 double 等类型的对象包装都是抽象类 Number 的子类。这就是说，任何需要一个 Number 实例的类，都可以向其传送一个 Integer、Long 、Float 或 Double 类。Number 类中的四个公共方法负责将一个数转换为一个特殊的原始数据类型：

```
public byte byteValue( )
public short shortValue( )
public abstract int intValue( )
public abstract long longValue( )
public abstract float floatValue( )
public abstract double doubleValue( )
```

47.17　Integer 类

Integer 类实现了对 int 数据类型的对象包装，它提供了转换整数到字符串及其相反操作的方法。我们可以从一个 int 值或一个字符串来创建一个 Integer 对象：

```
public Integer(int value)
public Integer(String s) throws NumberFormatException
```

当使用一个字符串创建整数时，Integer 类假设基(数字基)为 10，如果字符串中包含有非数字字符，则将产生一个 NumberFormatException 异常。与 Boolean 类中的 getBoolean( )方法相似，getInteger( )方法可转换一个来自系统属性的字符串。如果指定的属性值未设置，则返回 0 或作为第二个参数指定的缺省返回值。缺省值可以使用 int 或 Integer 的形式传送：

```
public statec Integer getInteger(String paramName)
public statec Integer getInteger(String paramName,int defaultValue)
public statec Integer getInteger(String paramName,Integer defaultValue)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Integer 类也提供了转换字符串为整数的方法，可以转换成 int，也可以转换成 Integer。我们也可以指定一个可选择的基(数字基)：

```
public static int parseInt(String s) throws NumberFormatException
public static int parseInt(String s,int radix)
    throws NumberFormatException
public static Integer valueOf(String s)throws NumberFormatException
public static Integer valueOf(String s,int radix)
    throws NumberFormatException
```

parseInt()与 valueOf()之间的唯一区别是，parseInt()返回一个 int，而 valueOf()返回一个 Integer。

许多情况下，我们需要在事先并不知道数字基的情况下，将一个字符串转换为一个数值。decode()方法可以区分开十进制数，十六进制数和八进制数：

```
public static Integer decode(String s) throws NumberFormatException
```

decode()方法通过检查数字的开始部分，可以分析出具体的数字基。如果以 0x 或 0X 开始，则认为是十六进制数；如果以 0 开始，则认为是八进制数；否则，认为是十进制数。

使用 toString()方法可将一个整数转换为一个字符串。toString()方法共有两个静态版本，不应将其与为 Object 的所有子类所定义的实例方法 toString() 相混淆。以一个 int 作为一个参数的静态方法，将 int 型的参数转换成一个字符串，在进行转换时也允许指定一个可选择的基。而实例方法 toString 将一个 Integer 实例的值转换为一个以 10 为基的字符表示：

```
public static String toString(int i)
public static String toString(int i,int radix)
```

最后，Integer.MIN_VALUE 和 Integer.MAX_VALUE 包含了 Java 中整数的最小和最大值：

```
public final static int MIN_VALUE
public final static int MAX_VALUE
```

47.18 Long 类

除了 Long 是 long 类型值的一个对象包装之外，其它方面它与 Integer 类是一样的。Long 的构造器为：

```
public Long(long value)
public Long(String s) throws NumberFormatException
```

使用 getLong()方法可以从系统属性中提取 Long 值：

```
public static Long getLong(String paramName)
public static Long getLong(String paramName, long defaultValue)
public static Long getLong(String paramName, Long defaultValue)
```

parseLong()和 valueOf()方法分别将一个字符串转换为 long 数据类型或 Long 对象：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public static Long parseLong(String s) throws NumberFormatException
public static Long parseLong(String s,int radix)
    throws NumberFormatException
public static Long valueOf(String s) throws NumberFormatException
public static Long valueOf(String s,int radix)
    throws NumberFormatException
```

toString()静态方法将 long 型数据类型转换为字符串：

```
public static String toString(long l)
public static String toString(long l,int radix)
```

最后，Long.MIN_VALUE 和 Long.MAX_VALUE 常量分别定义了长型数的最小数和最大数：

```
public final static long MIN_VALUE
public final static long MAX_VALUE
```

47.19　Byte 类

Byte 类与 Integer 类也十分相似。Byte 类有两个构造器：

```
public Byte(byte value)
public Byte(String s) throws NumberFormatException
```

与 Integer 和 Long 类不同的是，Byte 类不包含任何从系统属性中提取信息的方法。然而，我们可以将字符串分解成为字节数据。parseByte()和 valueOf()方法可分别将字符串转换成为 byte 数据类型和 Byte 对象：

```
public static byte parseByte(String s)
    throws NumberFormatException
public static byte parseByte(String s,int radix)
    throws NumberFormatException
public static Byte valueOf(String s) throws NumberFormatException
public static Byte valueOf(String s,int radix)
    throws NumberFormatException
```

另外，使用 decode()方法可以将一个十进制/十六进制/八进制字符串转换为一个 Byte 对象：

```
public static Byte decode(String s) throws NumberFormatException
```

toString()静态方法将 byte 数据类型转换为字符串：

```
public String toString()
public static String toString(byte b)
```

最后，Byte.MIN_VALUE 和 Byte.MAX_VALUE 常量分别定义 byte 数的最小值和最大值：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public final static byte MIN_VALUE
public final static byte MAX_VALUE
```

47.20 Short 类

到此，我们已经可以看出 Integer、Long、Byte 类的基本框架了。Short 类的框架也一样，即 Short 类支持以下方法：

```
public Short(short value)
public Short(String s) throws NumberFormatException
public static Short parseShort(String s) NumberFormatException
public static Short parseShort(String s,int radix)
    throws NumberFormatException
public static Short valueOf(String s) throws NumberFormatException
public static Short valueOf(String s,int radix)
    throws NumberFormatException
public static Short decode(String s) throws NumberFormatException
public String toString()
public static String toString(short s)
public final static short MIN_VALUE
public final static short MAX_VALUE
```

47.21 Float 类

Float 类为浮点型数据类型提供了一个对象包装。除了字符串转换功能之外，它还提供了直接操作一个 float 数的位的手段。可以通过给出一个 float 或 double 数，或一个数的字符串表示，创建一个 Float：

```
public Float(float value)
public Float(double value)
public Float(String s) throws NumberFormatException
```

Float 类中没有 Integer 和 Long 类中所提供的从系统属性中提取值的方法，但它提供了与字符串之间相互转换的方法。JDK 1.2 中还增加了与 parseInt()和 parseLong()方法等价的方法，以及一些其它相关的方法：

```
public static Float valueOf(String s)throws NumberFormatException
public static String toString(float f)
public static float parseFloat(String floatString)
```

浮点数有一个无穷大概念，还有一个“不是一个数(Not a Number，NaN)”的概念。我们可以使用 isInfinite()和 isNaN()方法测试这些值，这两个方法都有静态和实例两种变化形式：

```
public static boolean isInfinite(float f)
public static boolean isNaN(float f)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public boolean isInfinite()
public boolean isNaN()
```

如果要操作符点数的单个位，可使用 floatToIntBits()方法，将其转换为一个 int：

```
public static int floatToIntBits(float f)
```

float 和 double 的值均以 IEEE 745 的格式存储。该方法可能对我们并没有什么用处，除非是对这种存储格式非常熟悉，但是有的应用程序要使用这种信息。当在 int 的形式下对该数进行位操作之后，我们可以将其再转换为 float 型：

```
public static float intBitsToFloat(int bits)
```

应该记住，这种位操作表示与将一个 float 数转换成一个 int 数的情况不同。例如，Float.floatToIntBits((float)42)返回一个整数值 1109917696，这与其原始值有若干数量级的差别。

除了典型的 MIN_VALUE 和 MAX_VALUE 常量之外，Float 类还提供了 NEGATIVE_INFINITY、POSITIVE_INFINITY 以及“不是一个数(Not a Number)”或称为 NaN 的常量：

```
public final static float MIN_VALUE
public final static float MAX_VALUE
public final static float NEGATIVE_INFINITY
public final static float POSITIVE_INFINITY
public final static float NaN
```

47.22 Double 类

Double 类提供了与 Float 类相同的功能，只是它所处理的是 double 数据类型而不是 float 数据类型。使用一个 double 型数据或一个字符串，我们可以创建一个 Double：

```
public Double(double value)
public Double(String s) throws NumberFormatException
```

使用 toString()和 valueOf()方法，可以将一个 double 数与字符串相互转换：

```
public static String toString(double d)
public static double parseDouble(String doubleString)
public static Double valueOf(String s)
    throws NumberFormatException
```

Double 类也提供了检查无穷大和 NaN 的静态和实例版本的方法 isInfinite() 和 isNaN()：

```
public static boolean isInfinite(double d)
public static boolean isNaN(double d)
public boolean isInfitite()
public boolean isNaN()
```

使用 doubleToLongBits()和 LongBitsToDouble()方法，我们可以对一个双精度数进行位操

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

作,这些值的存储格式也是 IEEE 754 中规定的格式:

```
public static long doubleToLongBits(double d)
public static double LongBitsToDouble(long bits)
```

最后,Double 类也定义了其自己的 MIN_VALUE、MAX_VALUE、POSITIVE_INFINITY、NEGATIVE_INFINITY、以及 NaN 常量:

```
public final static double MIN_VALUE
public final static double MAX_VALUE
public final static double NEGATIVE_INFINITY
public final static double POSITIVE_INFINITY
public final static double NaN
```

47.23 Void 类

为了完整地包装原始数据类型,Sun 公司创建了一个 Void 类。由于 Void 类型不包含任何信息,我们可能会认为 Void 类中也没有构造器或方法。实际上正是这样,Void 类中所包含的唯一的东西是其它所有包装类都有的 TYPE 属性。

47.24 java.math.BigInteger 类

虽然 BigInteger 类也是 Number 的一个子类,但它并不是现有数据类型的包装类。相反,它实现了一套对于超大整数进行的算术操作,这些操作常用于加密应用程序,这种情况下所处理的数包括高达数百位或数千位。BigInteger 类实际上并不是 java.lang 软件包的一部分,而是与 BigDecimal 类一起都是 java.math 的一部分,BigDecimal 类实现了超大型的定点数。

47.24.1 创建 BigInteger

BigInteger 的值基本上想要多大就有多大。如果我们想要一个 927 位数字的数,那么我们就可以得到这样一个数。我们可以使用多种方式创建一个 BigInteger 对象。最简单的方法是从一个现有长型整数创建一个 BigInteger:

```
public BigInteger valueOf(long l)
```

> **注意**:valueOf()方法执行了与创建一个新的 BigInteger 实例的构造器相同的功能。由于它能够将现有的数如 0 和 1 作为常量 BigInteger 对象使用,所以它不是作为构造器实现的。

也可能使用一个字符串表示一个数,这在所创建的数很大以至于在 long 数据类型中存储不下时很有用:

```
public BigInteger(String str)
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

也可以使用一个字节数组来创建 BigInteger。

在这种情况下，字节数组实际上看起来像一个位数组。第一个字节中最左边的一位是该数中最重要的一位。切记这些字节并非数字的 ASCII 表示；它们包含了实际的数。我们可以使用下面的方法提取包含 BigInteger 表示的字节数组：

```
public byte[ ] toByteArray( )
```

在执行加密时，常常需要创建一个大型随机数，BigInteger 类具有产生这样一个数的功能。

在创建随机数时，bits 参数表示所要创建的数的位(bit)数，randomSource 对象用来产生随机位，所产生的数总是正数。

产生随机数时一个重要的因素是一个数为素数的概率，特别是在加密这一领域。我们可以使用一定的素数概率产生随机数：

```
public BigInteger(int bits,int certainty,Random randomSource)
```

其中的参数 certainty 指出了构造器产生一个素数的肯定度。概率由 1 - (1/(2^certainty))给出。肯定度 certainty 为 0 将产生一个 0 概率(2^0 = 1，所以上述公式为 1 - 1/1)，这意味着所产生的数可能不是素数。肯定度为 1 产生一个 0.5 的概率，肯定度为 10 将有一个 0.999(1 - 1/1024)的概率。对这一构造器，bits 参数必须至少为 2。

由于 Java 中不存在对大型数的内置支持，所以 BigInteger 类必须提供常用数值运算的方法。下面是可以使用的方法：

```
public BigInteger add(BigInteger otherValue)
public BigInteger subtract(BigInteger otherValue)
public BigInteger multiply(BigInteger otherValue)
public BigInteger divide(BigInteger otherValue)
public BigInteger remainder(BigInteger otherValue)
public BigInteger[ ] divideAndRemainder(BigInteger otherValue)
public BigInteger pow(int exponent)
public BigInteger gcd(BigInteger otherValue)
public BigInteger abs( )
public BigInteger negative( )
public BigInteger signum( )
public BigInteger mod(BigInteger modValue)
public BigInteger modPow(BigInteger exponent,BigInteger otherValue)
public BigInteger modInverse(BigInteger modValue)
public BigInteger shiftLeft(numBits)
public BigInteger shiftRight(numBits)
public BigInteger and(BigInteger otherValue)
public BigInteger or(BigInteger otherValue)
public BigInteger xor(BigInteger otherValue)
public BigInteger not( )
public BigInteger andNot(BigInteger otherValue)
public boolean testBit(int BitNumber)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public BigInteger setBit(int BitNumber)
public BigInteger clearBit(int BitNumber)
public BigInteger flipBit(int BitNumber)
public int getLowestSetBit()
public int bitLength()
public int bitCount()
public boolean isProbablePrime(int certainty)
public int compareTo(BigInteger otherValue)
public boolean equals(Object x)
public BigInteger min(BigInteger otherValue)
public BigInteger max(BigInteger otherValue)
```

最后,我们可以将一个 BigInteger 值转换为一个数值数据类型的值,但如果这个数太大,所要转换的数据类型容纳不下,则我们将丢失一些精度。

```
public int intValue()
public long longValue()
public float floatValue()
public double doubleValue()
```

47.25 java.math.BigDecimal 类

BigDecimal 类表示一个大型定点数。与 BigInteger 一样,它实际也是 Number 类的一个子类,而并非本地 Java 数据类型的对象包装。BigDecimal 数类似于 BigInteger 数,只是它有一个额外的度量参数,表示小数点右边的数字位数。

47.25.1 创建一个 BigDecimal

我们可以用一个双精度数或一个数字字符串来创建 BigDecimal:

```
public BigDecimal(double doubleValue)
public BigDecimal(String digits)
```

也可以从一个 BigInteger 来创建 BigDecimal。我们可以提供一个可选的 scale 参数用来指明小数点右边的数字个数。例如,一个 123456789 数,当度量数 scale 的值为 4 时,将产生一个数 12345.6789:

```
public BigDecimal(BigInteger bigVal)
public BigDecimal(BigInteger bigVal,int scale)
```

我们也可以使用 valueOf()方法,从一个 long 型值产生一个 BigDecimal:

```
public BigDecimal valueOf(long longValue)
public BigDecimal valueOf(long longValue,int scale)
```

在执行定点数计算时必须处理的一个问题是舍入问题。BigDecimal 类有几种舍入选择:

ROUND_DOWN	总是向下舍入
ROUND_HALF_UP	当最后一位大于等于 5 时向上舍入

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

ROUND_HALF_DOWN	当最后一位小于 5 时向下舍入
ROUND_UP	总是向上舍入
ROUND_CEILING	正数向上舍入,负数向下舍入
ROUND_FLOOR	正数向下舍入,负数向上舍入
ROUND_HALF_EVEN	如果小数点左边的一位为奇数,舍入方法与 ROUND_HALF_UP 相同;如果小数点左边的一位为偶数,舍入方法与 ROUND_HALF_DOWN 相同
ROUND_UNNECESSARY	总不舍入

这些舍入值只在进行除法运算和改变了数的度量值 scale 时使用。

与 BigInteger 类一样,BigDecimal 类也必须提供常用数值运算的方法。下面是可以使用的方法:

```
public BigDecimal add(BigDecimal otherValue)
public BigDecimal subtract(BigDecimal otherValue)
public BigDecimal multiply(BigDecimal otherValue)
public BigDecimal divide(BigDecimal otherValue, int roundingMode)
public BigDecimal divide(BigDecimal otherValue,
    int scale,int roundingMode)
public BigDecimal abs()
public BigDecimal negate()
public int signum()
public BigDecimal setScale(int scale)
public BigDecimal setScale(int scale,int roundingMode)
public BigDecimal movePointLeft(int numPositions)
public BigDecimal movePointRight(int numPositions)
public int compareTo(BigDecimal otherValue)
public boolean equals(Object x)
public BigDecimal min(BigDecimal otherValue)
public BigDecimal max(BigDecimal otherValue)
```

我们也可以将一个 BigDecimal 值转换为一个数值数据类型的值,但是如果这个数太大,所转换的数据类型无法容纳时,将会有精度丢失:

```
public int intValue()
public long longValue()
public float floatValue()
public double doubleValue()
public BigInteger toBigIntegr()
```

47.26　ClassLoader 类

ClassLoader 类包含了加载新的类到 Java 运行环境的方法。Java 最具威力的特性之一是其随时加载新类的能力。它与 Class 类一起使得动态类加载成为可能。ClassLoader 类中的所

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

有方法都是保护型的，这就是说它们只能被 ClassLoader 的子类访问。事实上除了一个方法之外，ClassLoader 的所有其它方法都是 final 型的，只有一个抽象的方法是通过单独的类加载器实现的。这个抽象方法是 loadClass()：

```
protected abstract Class loadClass(String className,boolean resolve)
    throws ClassNotFoundException
```

loadClass()方法用来从一个本地文件或通过网络查找类信息，并创建一个类。该方法首先为要加载的类获得一个表示 .class 文件全部内容的字节数组，然后调用 defineClass 方法，为新类创建一个 Class 实例：

```
protected final Class defineClass(byte data[],int offset,int length)
```

length 参数是定义类的字节数，offset 参数是类数据在 data 数组中的第一个字节的位置。

如果 loadClass()方法中的 resolve 参数为 true(真)，则 loadClass() 方法在返回之前负责调用 resolveClass()方法：

```
protected final void resolveClass(Class c)
```

resolveClass()方法确保指定的类 c 所引用的所有类都已经加载并解析。当一个类被解析后，它的类加载器将负责定位它所引用的任何其它类。例如，当一个类引用 java.lang.Object 时，这并不很方便。所以并没有强迫我们自己编写知道如何加载所有系统类的类加载器，相反，ClassLoader()类为我们提供了一个使用系统类加载器的中转，所以，如果我们不能定位一个类，那么在放弃之前可试着使用系统类加载器：

```
protected final Class findSystemClass(String name)
    throws ClassNotFoundException
```

程序清单 47.5 给出了一个类加载器的样本，该类加载器从另外一个目录下加载类。



```
import java.io.*;
import java.util.*;

// This class loader uses an alternate directory for loading classes.
// When a class is resolved, its class loader is expected to be able
// to load any additional classes, but this loader doesn't want to have
// to figure out where to find java.lang.Object, for instance, so it
// uses Class.forName to locate classes that the system already knows
// about.

public class MyClassLoader extends ClassLoader
{
    String classDir;        // root dir to load classes from
    Hashtable loadedClasses;        // Classes that have been loaded
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public MyClassLoader(String classDir)
    {
        this.classDir = classDir;
        loadedClasses = new Hashtable();
    }

    public synchronized Class loadClass(String className,
        boolean resolve) throws ClassNotFoundException
    {
        Class newClass = (Class) loadedClasses.get(className);

// If the class was in the loadedClasses table, you don't
// have to load it again, but you better resolve it, just
// in case.
        if (newClass != null)
        {
            if (resolve) // Should we resolve?
            {
                resolveClass(newClass);
            }
            return newClass;
        }

        try {
// Read in the class file
            byte[] classData = getClassData(className);
// Define the new class
            newClass = defineClass(classData, 0,classData.length);
        } catch (IOException readError) {

// Before you throw an exception, see if the system already knows
// about this class
            try {
                newClass = findSystemClass(className);
                return newClass;
            } catch (Exception any) {
                throw new ClassNotFoundException(className);
            }
        }

// Store the class in the table of loaded classes
        loadedClasses.put(className, newClass);

// If you are supposed to resolve this class, do it
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            if (resolve)
            {
                resolveClass(newClass);
            }

            return newClass;
        }

    // This version of loadClass uses classDir as the root directory
    // for where to look for classes, it then opens up a read stream
    // and reads in the class file as-is.

        protected byte[] getClassData(String className)
        throws IOException
        {
    // Rather than opening up a FileInputStream directly, you create
    // a File instance first so you can use the length method to
    // determine how big a buffer to allocate for the class

            File classFile = new File(classDir, className+".class");

            byte[] classData = new byte[(int)classFile.length()];
    // Now open up the input stream
            FileInputStream inFile = new FileInputStream(classFile);

    // Read in the class
            int length = inFile.read(classData);

            inFile.close();

            return classData;
        }
    }
```

程序清单47.6给出了一个用来测试加载器的简单类。

```
public class LoadMe extends Object
{
    public LoadMe()
    {
    }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public String toString()
    {
        return "Hello! This is the LoadMe object!";
    }
}
```

程序清单 47.7 中的 TestLoader 程序使用 MyClassLoader 加载 LoadMe 类，并将其打印出来。这里需要 LoadMe.class 文件存放在一个名为 TESTDIR 的子目录下。

程序清单 47.7　TestLoader.java 的源代码

```
//
// This program uses MyTestLoader to load the LoadMe class.
//

public class TestLoader extends Object
{
    public static void main(String[] args)
    {
// Create the class loader.  Note: myLoader must be declared as MyClassLoader
// and not ClassLoader because the loadClass method in ClassLoader is
// protected, not public.

        MyClassLoader myLoader = new MyClassLoader("testdir");

        try {
// Try to load the class
            Class loadMeClass = myLoader.loadClass("LoadMe", true);
// Create a new instance of the class
            Object loadMe = loadMeClass.newInstance();
// Print out the string representation of the instance
            System.out.println(loadMe);
        } catch (Exception oops) {
// If there was an error, just print a whole stack trace
            oops.printStackTrace();
        }
    }
}
```

47.27　SecurityManager 类

SecurityManager 类是 Java 安全性的关键类之一。它包含了一系列用来检查某一特殊操

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

作是否允许的方法。对于 Java 的各种系统类如 Applet 类、网络类及文件类等，在允许潜在的灾难性操作发生之前，都要使用安全性管理器检查。如果安全性管理器认为不允许进行某种操作，它将抛出一个 SecurityException 异常。System 类提供了访问当前安全性管理器及在尚未定义的情况下设置安全性管理的方法。当一个安全性管理器设置完之后，我们不能再对其进行修改。这些方法在 System 类中，而不是在 SecurityManager 类中：

```
public static SecurityManager getSecurityManager()
public static void setSecurityManager(SecurityManager)
    throws SecurityException
```

47.28 Compiler 类

如果 Java 只是一种解释性语言，那么在这个商品社会中它是不可能竞争过编译语言如 C++ 而存活下来的。JIT(Just - In - Time)编译器给予了 Java 它所需的速度。JIT 编译器在执行方法之前，将其编译为内部机器指令。这种编译对每个方法只进行一次。有些 JIT 编译器可能会一次编译完所有的类，而不是一次只编译一个类。Compiler 类可允许我们对 JIT 做少量的控制。

通过向 compileClass()方法传送一个 Class 实例，我们可以请求 JIT 编译一个指定的类：

```
public static boolean compileClass(Class clazz)
```

如果编译成功则 compileClass()方法返回 true(真)，如果编译失败或没有 JIT 可以使用则该方法返回 false(假)。如果我们需要激活一个方法，并且不希望在激活该方法时进行一次性编译，那么这个方法将是很有用的。我们可以请求 JIT 在开始激活方法之前预编译整个类：

compileClasses()方法与 compileClass()方法类似，只是它允许编译一组类：

```
public static boolean compileClasses(String classes)
```

classes 参数中包含了我们需要编译的类的名字。它可能是形如 java.lang.* 的一个字符串。读者可以查阅有关自己的 JIT(如果有的话)的文档，以获得关于这一方法更详细的介绍。

我们也可以使用 disable()和 enable()方法，有选择地禁用或选用 JIT 编译器：

```
public static void disable()
public static void enable()
```

最后，command()方法允许我们向编译器传送任意命令。这一方法是 JIT 相关的，对于特定的 JIT，我们应查阅其文档，找出它所支持的命令。Command() 方法格式为：

```
public static Object command(Object any)
```

- 如何捕获异常

有几种类型的异常，Java 坚持认为我们应该在自己的程序中进行处理。为了做到这一点，我们必须捕获到异常，然后执行一些动作。

- 创建和抛出自己的异常

Java 的 Exception 类允许我们创建自己定义的异常对象。我们可以在自己的程序中创建并抛出这些自定义的异常对象。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 关于 Java 的 Event(事件)类

当我们理解了 Event 类的工作方式后,也就为在自己的 Java 程序中处理事件做好了准备。

- 处理所有事件,包括非常重要的鼠标和键盘事件

我们的程序与用户进行交互的唯一途径就是通过事件。所以显然,处理事件是 Java 程序所必须的。

- 如何创建和发送自己的事件对象

有时候我们需要创建并发送自己的事件对象,以响应其它 Java 事件。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第48章 反　　射

本章内容：

48.1 什么是反射
48.2 创建一个只知道构造器列表的类
48.3 检查类的方法
48.4 使用 getDeclaredMethod( )方法激活方法
48.5 激活使用本地类型作为参数的方法
48.6 获得一个类的声明字段

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

48.1　什么是反射

按照 Sun 公司的解释，反射(Reflection)是一个“在当前 JVM 中支持类和对象内省的小型的、类型安全的和可靠的 API”。这需要做一些解释。基本上说，在这一定义中的关键字是内省(Introspection)。使用反射我们可以将一个对象如 Vector 拿到显微镜下面来进行观察，找出它所扩展的类和它所拥有的方法和变量，并且可以在不知道该对象是一个向量的情况下这样做。

为了完成内省，Sun 公司在 java.lang 包中添加了一组类，包括 Field、Method 及 Constructor。这些类都是用来从一个对象获取它们各自的特性信息的。另外，为了处理类的其余部分，Sun 增加了 Array 和 Modifier 类。

反射的全部用途都是为了增强 Java 的安全模型。安全模型防止那些对方法、字段、构造器等没有访问权限的类看到这些内容。安全模式使用反射所进行的工作是通过一些新类中的方法与安全性管理器的紧密结合来实现的。为了做到这一点，SecurityManager 本身增加了一个附加方法 checkMemberAccess( )。当要求一个类产生一个其自己的 Method 类时(注：只有类才允许创建 Method、Field 及 Constructor 类)，它首先查询 SecurityManager，确定是否可以给请求类一个其自己的 Method 类的拷贝。如果可以，那么不存在问题；如果不可以，那么请求被拒绝。这听起来好像是有人在牵强地使用类这个词，让我们这样看待这个问题：假设有这么一种情况，对象 Requestor 需要知道对象 Provider 的方法，在 Requestor 中，我们要知道 Provider 中可使用的构造器是什么。为做到这一点，我们可以象在程序清单 48.1 中所看到的那样创建两个类。

> **注意**：在 JDK 1.2 中，我们还可以通过扩展 AccessibleObject 或使用 ReflectPermission 类定义一个类的可访问性。

程序清单 48.1　Requestor 要从 Provider 请求方法

```
/*
*
* Requestor
*
*/
import java.lang.reflect.*;
public class Requestor{
    public void requestConstuctors(){
        try{
            Constructor con[] = Class.forName("Provider").getDeclaredConstructors();
            for (int x = 0;x < con.length;x++)
                System.out.println("Constructor "+x+" = "+con[x]);
        } catch (ClassNotFoundException se){
            System.out.println("Not allowed to get class info");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }

        public static void main(String args[]){
            Requestor req = new Requestor();
            req.requestConstuctors();
        }
}

/*
 *  Provider
 *
 */
class Provider{
        public Provider(){
        }

        public Provider(String s){
        }
}
```

当我们编译完这个类后，它应该称为 Requestor.java(注意我们必须使用 JDK 1.1 或 1.2 来编译它，在 1.0 下是不起作用的)，就可以运行 Requestor 程序了(也必须使用 JDK 1.1 或 1.2)。我们所得到的输出结果如下所示：

```
Constructor 0 = public Provider( )
Constructor 1 = public Provider(String)
```

这是一种很好的技巧，没有反射我们就根本无法实现这一点。在其它语言如 C/C++ 中，对于方法的访问可以使用方法指针来完成。由于 Java 没有指针，为了获得对运行时方法的访问，引入这一反射模型是很有必要的。

48.2 创建一个只知道构造器列表的类

让我们来看看其它的事情，使用 Provider(String)方法，实例化 Provider 类。在 JDK 1.0 下，这是不可能的(当然没有使用直接的手段)。程序清单 48.2 给出了完成这一任务的代码。注意，我们在创建时使用了 null 作为 newInstance( )方法的参数(可以这样做的原因是我们知道参数(String)有一个空(null)构造器)。

程序清单 48.2 在不知道构造器名字的情况下 Requestor 创建了一个 Provider

```
/*
 *
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
 * Requestor
 *
 */
import java.lang.reflect.*;
public class Requestor{
    public void requestConstuctors(){
        Class cl;
        Constructor con[];
        try{
            cl = Class.forName("Provider");
            con = cl.getDeclaredConstructors();
            for (int x = 0;x < con.length;x++)
                System.out.println("Constructor " - x + " = " + con[x]);
            Class param[] = con[1].getParameterTypes();
            Object paramValues[] = new Object[param.length];
            for (int x = 0;x < param.length;x++){
                if (! param[x].isPrimitive()){
                    System.out.println("param:" + param[x]);
                    paramValues[x] = param[x].newInstance();
                }
            }
            Object prov = con[1].newInstance(paramValues
        } catch (InvocationTargetException e){
        System.out.println("There was an InvocationEception and
➡ we were not allowed to get class info: " + e.getTargetException());
        } catch (Exception e){
        System.out.println("Exception during construction:" + e);
        }
    }

    public static void main(String args[]){
        Requestor req = new Requestor();
        req.requestConstuctors();
    }
}

class Provider{
    String me;
    public Provider(){
        me = new String();
    }

    public Provider(String x){
        this.me = me;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }
}
```

当然这个完整的系统是可以工作的，但可能不是很实用。总之所有我们需要做的只是构造一个缺省对象，并且这个对象没有其自己的空(null)构造器，这种情况是不常见的。当出现这种情况时更有可能的是需要实例化一个具有我们所希望的构造器的对象。例如，如果我们要建造一个 API，一个类的基础是带有多个参数的一个构造器，扩展基类的每个类必须重载这个构造器。这看起来太简单了，是这样吗？例如，让我们看看大家所熟悉的轿车模型，此处将有两个类 Car 和 Tires(见程序清单 48.3)。

程序清单 48.3 使用 Tires 类的 Car 类

```
public class Tires {
    int number;
    float diameter;
}

public class Car{
    Tires tires;
    public Car (Tires tires){
        this.tires = tires;
    }
}
```

没有必要写出这个类的其余部分了，因为它与现在我们所讨论的问题无关。问题是 Car 类显然需要从外部源接收一个 Tires 对象。当我们创建一个子类时，比如说 Saturn 和 BMW，我们仍需从外部源获得 Tires 模型，参见程序清单 48.4。

程序清单 48.4 使用 Tires 类的 Car 类的子类 Saturn 和 BMW

```
public class Saturn extends Car{
    public Saturn(Tires tires){
        super(tires);
    }
}

public class BMW extends Car{
    public BMW(Tires tires){
        super(tires);
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

现在的唯一问题是，在 JDK 1.0 下，至少要在程序中编码实现构造器，否则我们就没有真正的面向对象的手段来处理每个新轿车类型。我们可能会认为这听起来像是一个接口方案，但不幸的是接口的面很窄，不能处理这种情况。接口允许我们创建方法模板，但不允许创建构造器模板。这就是说每次引入一个新轿车时，我们必须回过头来找出新轿车需要添加的每个实例，并一一地手工添加。现在使用反射后一切都变得非常简单了，见程序清单 48.5。

程序清单 48.5　使用反射，完整的 CarShop 创建 Cars

```
/*
 *
 * CarShop
 *
 */
import java.lang.reflect.*;

public class CarShop {
    Car carList[];

    public CarShop (){
        carList = new Car[2];
        createCar("Saturn",0);
        createCar("BMW",1);
    }

    public void createCar(String carName,int carNum){
        try{
            //create the Tires array, which you'll use as a
            //the parameter to the constructor
            Object constructorParam[] = new Tires[1];
            constructorParam[0] = new Tires();

            //get the class name for the car that you want
            Class cl = Class.forName(carName);

            //create an array of Classes, and use this to
            //array to find the constructor that you want
            Class parameters[] = new Class[1];
            parameters[0] = Class.forName("Tires");
            Constructor con = cl.getDeclaredConstructor(parameters);

            //create a car instance for the carList
            carList[carNum] = (Car)con.newInstance(constructorParam);
        } catch (Exception e){
            System.out.println("Error creating " + carName + ":" + e);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
    }

    public static void main(String args[]){
        new CarShop();
    }
}
```

在这个例子中，最重要的显然是 createCar( )方法。现在我们将其一步步分解如下：

```
Object constructorParam[] = new Tires[1];
constructorParam[carNum] = new Tires( );
```

正如前面使用反射时所看到的一样，方法 newInstance 允许我们创建一个类的新实例，并以一个对象数组(Object[ ])作为其参数。作为结果，看一下我们所希望使用的构造器：

```
Car(Tires tire)
```

这个构造器有一个参数，其类型为 Tires，所以我们需要以 Tire 对象作为第一个(只能是第一)元素，创建一个数组：

```
Class cl = Class.forName(carName);
```

下一步，我们需要获得所要寻找的类。注意，我们只要知道类的名字就可以做到这一点。这意思是说我们甚至可以创建一个 < PARAM > 值，它包含所有当前所知道的轿车的一个清单：

```
Class parameters[] = new Class[1];
parameters[0] = Class.forName("Tires");
Constructor con = cl.getDeclaredConstructor(parameters);
```

再下一步是找出与我们正在寻找的构造器相匹配的构造器。为此，我们必须创建一个类数组。这个数组的每个元素是与所需查找的构造器参数顺序匹配的一个类：

```
carList[carNum] = (Car)con.newInstance(constructorParam);
```

现在，我们已经有了一个正确的构造器，并把参数数组放在了一起，我们可以最终创建新的轿车实例了。我们可能会自己说“所有这些不也就是 carList[carNum] = new Saturn(new Tire( ));吗?”

是的，确实如此，但是为了真正说明这一情况，我们还需使用一个 if 语句，如下所示：

```
if(carName.equals("Saturn"))
    carList[carNum] = new Saturn(new Tire( ));
else if(carName.equals("BMW"))
    carList[carNum] = new BMW(new Tire( ));
```

每次添加一个新车时，我们必须回来再加上一个 if 语句。而使用反射则不必。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

48.3 检查类的方法

除了可以找出类方法之外，反射还能帮助我们找出一个类中所具有的方法。为了做到这一点，需要创建一个类似于 getDeclaredConstructors( ) 的方法称为 getDeclaredMethods( )。

48.3.1 获得方法列表

回过头来看一下程序清单 48.1 中的 Requestor/Provider 示例，除了列出构造器列表之外，我们将向 Provider 类增加两个方法，看看当我们在 Requestor 类中发出请求时会发生什么，见程序清单 48.6。

程序清单 48.6 反射揭示出方法和构造器

```
/*
*
* Requestor
*
*/
import java.lang.reflect.*;

public class Requestor{
    public void requestConstuctors(){
        Class cl;
        Constructor con[];
        Method meth[];
        try{
            cl = Class.forName("Provider");
            con = cl.getDeclaredConstructors();
            for (int x = 0;x < con.length;x++)
                System.out.println("Constructor " + x + " = " + con[x]);

            meth = cl.getDeclaredMethods();
            for (int x = 0;x < meth.length;x++)
                System.out.println("Method " + x + " = " + meth[x]);

        } catch (NoSuchMethodException e){
            System.out.println(
        ➡"There was an exception and we were not allowed to get class info: " + e);

}
    }

    public static void main(String args[]){
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        Requestor req = new Requestor();
        req.requestConstuctors();
    }
}

class Provider{
    int x;
    public Provider(){
        this.x = 0;
    }

    public Provider(int x){
        this.x = x;
    }

    public boolean testMe(boolean test){
        return ! test;
    }

    public int addThree(int num){
        return num + 3;
    }

    public char letterD(){
        return 'D';
    }

}
```

现在，编译 Requestor.java，然后运行它，其输出结果如下所示：

```
Constructor 0 = public Provider( )
Constructor 1 = public Provider(int)
Method 0 = public boolean Provider.testMe(boolean)
Method 1 = public int Provider.addThree(int)
Method 2 = public char Provider.letterD( )
```

可以看出，该方法不仅像构造器时所做的那样包含了修饰符和参数，而且还有返回类型。我们不应为此感到惊讶，因为对任何一个方法这都是其关键组件。下面对 Provider 类进行一些修改，如程序清单 48.7 所示。

```
/x
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
 *
 * Provider
 *
 */

class Provider extends java.applet.Applet{
        int x;
        public Provider(){
                this.x = 0;
        }

        public Provider(int x){
                this.x = x;
        }

        public boolean testMe(boolean test){
                return ! test;
        }

        public int addThree(int num){
                return num + 3;
        }

        public char letterD(){
                return 'D';
        }

}
```

现在，再次运行 Requestor，结果如下所示：

```
Constructor 0 = public Provider( )
Constructor 1 = public Provider(int)
Method 0 = public boolean Provider.testMe(boolean)
Method 1 = public int Provider.addThree(int)
Method 2 = public char Provider.letterD( )
```

它们有什么区别呢？没有，它们之间根本没有什么不同。虽然实际上 Provider 是从 java.applet.Applet 继承而来，java.applet.Applet 本身有大量方法，但是这些方法并不显示在来自于 getDeclaredMethods()的列表中，这是因为 getDeclaredMethods( )只返回当前类(或接口)中声明的所有方法，而不返回该类通过继承而获得的方法。

也许我们会问“这意思是说如果我们覆盖或重载了一个方法，我们将不能检测到这个方法，因为它是通过继承获得的?”对于这两种情况答案都是否定的，我们是能够看到这些方法的。重载方法实际上是一个新方法，所以它们并非通过继承而来；而覆盖方法被包含在这个方

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

法列表中是为了避免混乱。

48.4 使用 getDeclaredMethod( )方法激活方法

正如我们所猜测的,就像构造器的例子一样,激活一个方法如果只是为了激活它的话,除了少数的实例如编写一个调试程序之外,实际上没有什么用处。

与构造器方法 getDeclaredConstructors( )相对应,getDeclaredMethods( )也有一个同胞兄弟 getDeclaredMethod(),该方法只获得我们所要查找的指定方法:

```
public Method getDeclaredMethod(String Name, Class parameterType[])
```

可以看出,getDeclaredMethod( )方法使用了一个附加参数。与构造器时的情况一样,这是因为查找一个方法时需要基于方法的参数列表进行查找。但是在查找一个构造器时,没有必要知道构造器的名字(name),因为每个构造器的名字是与类的名字相同的。在查找方法时,则需要同时指定方法的名字。

与和它对应的构造器方法一样,如果没有方法与我们所要查找的方法签名匹配,getDeclaredMethod 将抛出一个 NoSuchMethodException 异常。当然,这就是说我们需要把 getDeclaredMethod( )方法放在一个 try - catch 块中。

现在让我们回到轿车示例中并做一些修改,如程序清单 48.8 所示。

程序清单 48.8 修改后的 Car 类

```
class Car{
    Tires tires;
    boolean running = false;

    public Car (Tires tires){
        this.tires = tires;
    }
}


class Saturn extends Car{
    public Saturn(Tires tires){
        super(tires);
    }

    public boolean start(){
        running = true;
        System.out.println("The Saturn is now running");
        return true;
    }
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
class BMW extends Car{
    public BMW(Tires tires){
        super(tires);
    }

    public boolean start(){
        running = true;
        System.out.println("The BMW is now running");
        return true;
    }
}
```

首先，修改 Car 类，并向 Saturn 和 BMW 轿车增加一个 start( ) 方法。不要将 start( )方法增加到 Car 中，这样就太简单了。然后我们需要向 CarShop 类增加一个方法，允许它启动轿车。在这种情况下，调用 startCar( )方法，并且在将 Saturn 和 BMW 添加到汽车池中后立即调用。见程序清单 48.9。

程序清单 48.9　使用新的 CarShop

```
/*
*
* CarShop
*
*/
import java.lang.reflect.*;

public class CarShop {
    Car carList[];

    public CarShop (){
        carList = new Car[2];
        createCar("Saturn",0);
        createCar("BMW",1);
        startCar(1);
    }

    public void createCar(String carName,int carNum){
        try{
            //create the Tires array, that you'll use as a
            //the parameter to the constructor
            Object constructorParam[] = new Tires[1];
            constructorParam[0] = new Tires();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            //get the class name for the car that you want
            Class cl = Class.forName(carName);

            //create an array of Classes, and use this to
            //array to find the constructor that you want
            Class parameters[] = new Class[1];
            parameters[0] = Class.forName("Tires");
            Constructor con = cl.getDeclaredConstructor(parameters);

            //create a car instance for the carList
            carList[carNum] = (Car)con.newInstance(constructorParam);
        } catch (Exception e){
            System.out.println("Error creating " + carName +":" + e);
        }
    }

    public void startCar(int num){
        try{
            //create an array of Classes, and use this to
            //array to find the method you want
            //since you are actually looking for a null parameter
            //this is an array of 0
            Class parameters[] = new Class[0];
            Class carType = carList[num].getClass();
            Method meth = carType.getDeclaredMethod("start",parameters);

            //create a car instance for the carList
            meth.invoke(carList[num],parameters);
        } catch (Exception e){
            System.out.println("Error starting car " + num +":" + e);
        }
    }

    public static void main(String args[]){
        new CarShop();
    }
}
```

现在，当我们运行该应用程序时，它将通知我们 BMW 正在运行。关于 invoke( )方法必须强调几点。invoke( )方法需要两个参数，第二个参数是一个参数数组，用来激活方法，正如构造器的 newInstance( ) 方法中使用参数数组一样。然而，invoke( )方法也需要知道方法正在被哪个对象调用，所以 invoke( )的第一个参数是正确的对象。如果该对象没有 start( )方法会发生什么呢？还要比想象的更糟糕。如果该对象不是声明该方法的类的实例，将抛出一个

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

异常。

让我们再次回到程序清单48.9,注意一下为获得start( )方法,我们需要对BMW或Saturn类进行操作,而不是对这些类的对象实例进行操作,java.lang.Object有一个新方法称为getClass( ),它可以帮助我们很容易地解决这一问题,java.lang.Object也因此得到赞誉。

在这一点上,一个好的面向对象的程序员或许会问,"为什么我们要使用这一方法激活一个方法呢?"毕竟,一个更好的设计方案是在car( )类中使用一个接口或一个start( )方法。确实如此,世界上的一切都不会十全十美。如果没有像反射这样激活方法的能力,编程体系就会存在相当大的限制。更为重要的是,方法getDeclaredMethod( )经常通过一个方法指针提供另外一个方法。这就是说,在方法签名中变得更为重要的部分将成为参数列表,而不是方法的名字。这种层次的扩展,允许我们创建需要相似处理的多个方法,而不需创建多个处理层。

48.5 激活使用本地类型作为参数的方法

到目前为止,我们所看到的系统中存在一个限制,即我们还没有看到如何激活一个使用本地类型作为其参数的方法或构造器。例如,考虑如下的程序清单48.10中的NewCar。

程序清单48.10 NewCar有一个需要使用int作为参数的构造器

```
public class NewCar{
  int numTires;
  public NewCar(int numTires){
    this.numTires = numTires;
  }

  public int getTireCount(){
    return numTires;
  }
}
```

为激活NewCar的构造器,必须提供一个int。然而,getDeclaredConstructor()方法需要一个类对象数组。我们如何获取int类呢?为了处理这一特殊情况,语言允许我们为每种本地类型增加一个.Class,获得表示该类型的一个类对象。

下一个问题是提供一个表示本地类型的对象。我们可能已经猜测到,需要使用java.lang中的包装类,在这个例子中我们需要使用java.lang.Integer。下面的程序清单48.11给出了实例化NewCar的一个简单例子。

程序清单48.11 TestNewCar创建了NewCar类的一个新实例

```
import java.lang.reflect.Constructor;

public class TestNewCar{
  public static void main(String args[]){
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
        try{
          Class car = Class.forName("NewCar");

          Class param[] = {int.class};

          Constructor con = car.getDeclaredConstructor(param);

          Object values[] = {new Integer(4)};

          Object carObj = con.newInstance(values);

          System.out.println("The car has " + ((NewCar)carObj).getTireCount() + " tires");
        }catch (Exception e){
          System.out.println("Something went wrong while invoking NewCar");
          e.printStackTrace(System.err);
        }
    }
}
```

48.6 获得一个类的声明字段

本章中所介绍的反射的最后一个方面是获取一个类所包含的字段列表。可以猜测出，此处我们将用到两个最有用的方法，即 getDeclaredFields() 和 getDeclaredField()。在前面的 Provider 类中(程序清单 48.7)，我们已经有了一个变量(x)，所以我们现在需要做的只是对 Requestor 类做进一步改进，见程序清单 48.12。

程序清单 48.12 获取字段的 Requestor 应用程序

```
/*
 *
 * Requestor
 *
 */
import java.lang.reflect.*;

public class Requestor{
    public void requestConstuctors(){
        Class cl;
        Constructor con[];
        Method meth[];
        Field  field[];
        try{
            cl = Class.forName("Provider");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            con = cl.getDeclaredConstructors();
            for (int x = 0; x < con.length; x++)
                System.out.println("Constructor " + x + " = " + con[x]);

            meth = cl.getDeclaredMethods();
            for (int x = 0; x < meth.length; x++)
                System.out.println("Fethod " + x + " = " + meth[x]);

            field = cl.getDeclaredFields();
            for (int x = 0; x < field.length; x++)
                System.out.println("Tield " + x + " = " + field[x]);


        } catch (Exception e) {
            System.out.println("There was an exception and you were
➥not allowed to get class info: " + e);
        }
    }

    public static void main(String args[]) {
        Requestor req = new Requestor();
        req.requestConstuctors();
    }
}
```

如果看过了上面的两个例子，我们可能已经猜测出新的 Requestor 的输出结果了，如下所示：

```
Costructor 0 = public Provider( )
Costructor 1 = public Provider(int)
Method 0 = public boolean Provider.testMe (boolean)
Method 1 = public int Provider.addThree(int)
Method 2 = public char Provider.letterD( )
Method 3 = public boolean Provider.mouseDown(java.awt.Event,int,int)
Field 0 = int Provider.x
```

Field()本身可用于提供对 Field 类型的变宽或变窄转换，这部分内容留给读者作进一步探讨。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 49 章　使用其它语言扩展 Java

本章内容：

49.1　本地方法，Java 最终边界
49.2　本地化
49.3　JNI 精粹
49.4　编写本地方法
49.5　从本地方法访问对象字段
49.6　从本地方法访问 Java 方法
49.7　访问静态字段
49.8　访问静态方法
49.9　本地方法的异常处理

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

49.1　本地方法，Java 最终边界

本地方法这一问题曾经被称作为 Java 的“最终边界”之一。确实如此！这只是因为这种潜在的特性使得 Java 开发人员能够完成在计算机上可编程实现的任何事情——当然这里假设对于所要完成的事情存在 Java 实现。本地方法可使程序员在问题不能单独使用 Java 得到解决的情况下，通过有选择地使用其它编程语言如 C 或 C++ 实现一些方法，对 Java 的功能进行扩展。

在 JDK 1.1 之前，本地方法是那些可以预见的、很少讨论并且也很少能理解的 Java“高深科学”论题之一。这样说也并不是没有充足的理由。JVM 中很多本地方法接口的实现，根据实现它的开发商不同，存在着巨大的差别。由于缺点太多，JDK 1.02 下 Sun 公司的本地方法规范被其它专用的接口如 Netscape 的 Java Runtime Interface(JRI)、微软的 Raw Native Interface(RNI)以及 Java/COM 接口等所取代。缺少统一的接口意味着开发人员必须为每个所使用的 JVM 实现独立的本地方法，即使是它们都有相同的目标平台。

知道了这些主要情况，对于 JDK 1.1 下的本地方法规范所进行的彻底修改就不足为奇了。新的改进版本称为 Java Native Interface(JNI)，它通过为本地方法提供统一的接口简化了要做的事情，它不考虑 JVM 的使用，并假设开发商为支持本地方法遵守 JNI 规范。JNI 的开发是经过 Sun Microsystems 公司与其它 Java 许可商广泛协商后的结果，它与 Netscape 的 JRI 规范非常相似。

> **注意**：至于微软公司是否在其 JVM 实现中采用 JNI 标准尚不清楚。到目前为止，微软一直在坚持它将不采用 JNI，而只支持 RNI 和 Java/COM 接口。这实际上是我们可能听说过的 Sun 公司与微软公司之间的诉讼案的一部分。将来究竟会怎样，希望到 1998 年年底能有更好的说法。

49.2　本地化

Java 完美主义者可能会认为将 C 或 C++ 代码加入到 Java 程序中是一种异端行为。Java 与平台无关的 JDK 具有很强大的功能和灵活性，到底为什么还有人要使用 Java 以外的其它语言编程呢？原因有许多：

- Java 程序员可能需要访问特殊的操作系统工具，而这种工具使用 JDK 无法直接获得。
- 我们可能需要直接访问一些外围设备如视频卡、声卡或调制解调器等，以便能更好地使用它们所提供的功能。
- Java 程序可能需要与第三方中间件和信息系统(如 IBM 的 MQ Series、Sybase 的 Open Server、Momemtum 的 XIPC、Lotus 的 Notes 等)或其它专用企业级软件系统如 SAP R/3、PeopleSoft、Baan 等等进行接口。由于这些软件方案都带有其自己(通常是基于 C)的专用接口 API，本地方法就成了它们与 Java 集成的唯一选择。
- 大量遗留代码已经存在于后端，这些代码将继续平滑地工作下去，而不考虑用户接口。遗留系统在没有进行大量代码重写的情况下，继续移植到基于 Java 的 intranet/Internet 计算机环境下，这也是其使用本地方法迅速集成的主要原因。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

● 在实时 Java 系统中有些高度面向时间的操作可能需要它的有些功能以汇编代码实现。在这样的情况下，使用本地方法通常是唯一的解决方法。

加入本地方法会带来一定的好处，但肯定也会付出一些代价：

● 我们的 Java 代码失去了其主要优点之一：可移植性。由于本地方法的实现是平台专有的，对于每一个将要运行 Java 系统的平台，本地库都必须重新实现和重新编译。

● 浏览器安全性管理器策略可能会阻止我们加载那些连接于由本地方法实现的、动态可加载库的 Java 小程序。结果是本地方法只能在 Java 应用程序中自由地使用，而在 Java 小程序中则不能。

> **注意**：浏览器的未来版本可能会有更好更完整的自定制安全性管理器，到那时用户就可以在一个可选的基本框架下加载实现本地方法的 Java 小程序了。Sun 公司的 HotJava 浏览器目前提供了一个自定制安全性管理器。

● 本地方法的使用包括对附加头文件和 C 接口的管理。创建复杂的本地方法本身就不仅需要大量的 Java 编程专门知识，而且还需要实现语言和平台的专门知识。

49.3 JNI 精粹

正如前面所提到的，JVM 完全遵照 JDK 1.1 规范，提供了独立于平台的相同标准的本地方法接口，即 JNI。JNI 的主要内容如下：

- 本地方法可以创建、更新并检查 Java 对象。
- Java 可以以参数形式向本地方法传送任何原始数据类型或对象。
- 本地方法可以向 Java 环境返回原始的数据类型或对象。
- Java 实例或类方法可以在本地方法中进行调用。
- 本地方法可以捕获和抛出 Java 异常。
- 本地方法中可以进行运行时类型检查。
- 本地方法可以实现同步以支持多线程访问。

49.4 编写本地方法

如果认为本地方法是必经之路，那么现在就让我们进入 JNI 的神秘世界。

> **注意**：此处所给出的所有 JNI 例子都假设在 Solaris 操作环境下，并且本地方法使用的是 C 语言。这些例子也应该可以很好的运行于其它平台，只要正确地为相应平台创建共享库。

JNI 中所有本地方法的实现严格遵循以下的六个基本步骤。

49.4.1 第一步：编写 Java 代码

请看程序清单 49.1，我们可以看到，在 greet()方法的声明中有一个关键字 native，这说明该方法实现于 Java 外部，并且使用了一种不同的编程语言。另外，静态块告诉 Java 加载共享

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

库(libsayhello1.so,在 Solaris 环境下),运行时可以在共享库中找到该方法的具体实现。

程序清单 49.1　SayHello1.java——本地方法演示程序

```
public class SayHello1 {
    public native void greet();

    static {
    System.loadLibrary("sayhello1");
    }
        public static void main(String[] args) {
        new SayHello1().greet();
    }
}
```

其中 main 方法只创建 SayHello1 类的一个新实例,并激活了本地方法 greet。注意,本地方法在激活时与普通的实例方法完全一样。

49.4.2　第二步:将 Java 代码编译成类文件

这一步没有什么奇特之处,只是像通常情况一样使用 javac 编译器编译 Java 源文件:

```
javac SayHello1.java
```

49.4.3　第三步:产生 JNI 风格的头文件

对编译过的类文件使用 javah——作为 JDK 的一部分提供的用来生成 C 头文件和主干文件的生成器——产生类的 JNI 头文件。与 JDK 1.02 下的本地方法接口规范不同,JNI 不使用主干文件。

```
javah -jni SayHello1
```

这样就在本地目录生成一个相应的头文件 SayHello1.h(见程序清单 49.2)。

程序清单 49.2　SayHello1.h——所生成的头文件

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class SayHello1 */

#ifndef _Included_SayHello1
#define _Included_SayHello1
#ifdef __cplusplus
extern "C" {
#endif
/*
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
 * Class:     SayHello1
 * Method:    greet
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_SayHello1_greet
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

注意:所生成的头文件总是有一个形如 SomeClassFile.h 的命名习惯,在此处我们的 Java 源文件中,类文件 SomeClassFile 包含了本地方法声明。

49.4.4 第四步:实现本地方法

当确信本地方法实现文件中包含了所生成的头文件之后,就可以开发本地方法了。为了使问题简化,本地方法原型可以从头文件中获得(见程序清单 49.3)。

程序清单 49.3 MyHello1.C——本地方法实现程序

```
#include "SayHello1.h"
#include <stdio.h>
JNIEXPORT # void JNICALL # Java_SayHello1_greet (JNIEnv * env, jobject this) {
    printf("Hi folks! Welcome to the netherworld of native methods! \n");
}
```

提示:关于本地方法实现文件的命名没有什么限制,但是为方法所选择的文件名最好与实际声明本地方法的 Java 文件能有某种联系。

49.4.5 第五步:创建共享库

这一步将会因为目标平台的不同而有着很大的差别,它需要一个可以用来创建共享库的编译器。对 Solaris,可以通过使用适当的编译器选项很容易地获得一个共享对象。标准 JDK 1.2 的路径中应包含那些在编译过程中被适当注释的文件,这一点很重要。

例如,在 Solaris 下,下述命令生成一个共享对象 libsayhello1.so:

```
cc -G MyHello1.c -I $JAVAHOME/include -I $JAVAHOME/include/solaris
   -o libsayhello1.so
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

对于 Windows 95 和 Windows NT,可以使用如下命令为本地实现创建一个 DLL 文件:

```
cl MyHello1.c -I $ JAVAHOME \ include -I S JAVAHOME \ include \ win32
    -Fesayhello1.dll -MD -LD -nologo
S JAVAHOME \ lib \ javai.lib
```

注意:对于 Windows 95 和 Windows NT,可以使用任何能够产生 DLL 文件的 C 或 C++ 编译器。在创建 DLL 时我们必须确保包含相关 JDK,包括文件的路径。

49.4.6　第六步:运行 Java 程序

如下所示运行应用程序:

```
java SayHello1
```

所产生的输出结果如下:

```
Hi folks! Welcome to the netherworld of native methods!
```

提示:在 Solaris 平台上,要注意环境变量 LD _ LIBRARY _ PATH 中应当包含新创建的共享对象的路径。

49.5　从本地方法访问对象字段

JNI 方法可以很容易地访问甚至修改所调用对象的成员字段。这可以通过使用各种 JNI 访问器函数而实现,对程序员来说,这些访问器函数可以通过缺省情况下作为第一个参数传送给每一个 JNI 方法的接口指针而获得。JNI 接口指针的类型为 JNIEnv。第二个参数随本地方法实质的不同而不同。对于非静态本地方法,该参数是对当前对象的引用;而对于静态方法,该参数是对其 Java 类的引用。

通过访问器函数访问 Java 对象的成员保证了本地方法实现的可移植性。假设一个 VM 的开发商实现了 JNI,那么我们的本地方法应该是可以运行的,不管 Java 对象是如何进行内部维护的。

让我们来看看程序清单 49.3 中这种函数的原型:

```
JNIEXPORT # void JNICALL # Java _ SayHello1 _ greet(JNIEnv * env, jobject this)
```

可以看出,缺省时这一方法的前两个参数是由 Java 环境传送的。接口指针 env 可使我们对访问器函数进行访问,对象引用 this 指的是激活本地方法的实例。

程序清单 49.4 是程序清单 49.1 的修改版,我们向类中增加了两个公共成员字段,可以从程序清单 49.5 中看出如何访问和修改这些成员。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 49.4 SayHello2.java——从本地方法中访问对象的演示 Java 程序

```
public class SayHello2 {

        public String aPal = "Java Joe";
        public int age = 0;
        public native void greet();


        static {
            System.loadLibrary("sayhello2");
        }

        public void howOld() {
            System.out.println(aPal + "is" + age + "years old today!");
        }
        public static void main(String[] args) {
            SayHello2 mySayHello2 = new SayHello2();
             mySayHello2.greet();
             mySayHello2.howOld();
        }
}
```

程序清单 49.5 SayHello2.c——演示对象字段的本地方法实现

```
#include "SayHello2.h"
#include <studio.h>
JNIEXPORT # void JNICALL # Java_SayHello2_greet ( JNIEnv *env, jobject this) {
    jfieldID jf;
    jclass jc;
    jobject jobj;
    const jbyte *pal;
       jint new_age = 2;
       jc = (*env) -> GetObjectClass(env, this);
       jf = (*env) -> GetFieldID(env,jc,"aPal","Ljava/lang/String;");
       jobj = (*env) -> GetObjectField(env,this,jf);
       pal = (*env) -> GetStringUTFChars(env,jobj,0);
       printf("Hi %s! Welcome to the netherworld of native methods! \n", pal);
       jf = (*env) -> GetFieldID(env,jc,"age","I");
       (*env) -> SetIntField(env,this,jf,new_age);
}
```

在程序清单 49.5 中，我们可以看到如何在本地方法中访问字段的值，以及如何重新设置

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java 字段的值。

Java 的类文件和共享库与前面相同，执行后其输出结果如下：

```
Hi Java Joe! Welcome to the netherworld of native mothods!
Java Joe is 2 years old today!
```

> 错误检查：
>
> 我们为什么获得了所有的异常错误？
>
> 如果我们遇到大量的 Java 异常错误，最有可能的原因是我们的 getMethodID( )方法调用中使用了不正确的类型签名。检查类型签名是否正确，并重新编译共享库。

在本地方法内访问和使用对象字段应遵照如下四个步骤：

1. 获得所激活对象的类的类型。
2. 获得字段 ID。
3. 使用适当的 GetField( )/SetField( )访问器函数检索/设置对象字段。
4. 根据需要转换所检索到的对象字段以便在本地方法中使用。

getObjectClass 用来确定激活本地方法的对象的类。可以看出，在访问对象成员字段的值之前，我们首先需要获取该字段的 fieldID。GetFieldID 访问器函数需要确切知道字段名字以及其类型签名。表 49.1 给出了所有可能的 JVM 类型签名。

表 49.1　JVM 类型签名

类型签名	Java 类型
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L 完整说明的类	完整说明的类
[类型	类型[]
(参数类型)返回类型	方法类型

由于我们的例子中所访问的字段类型为 String，其类型签名为 Ljava/lang/String。JNI 接口指针根据其类型的不同提供了许多访问实际成员字段的函数。

表 49.2 给出了各种可以使用的 GetField( )函数。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 49.2 用于 Java 字段访问的访问器函数

GetField 例程名	本地类型	Java 类型
GetObjectField( )	jobject	Object
GetBooleanField( )	jboolean	boolean
GetByteField( )	jbyte	byte
GetCharField( )	jchar	char
GetShortField( )	jshort	short
GetIntField( )	jint	int
GetLongField( )	jlong	long
GetFloatField( )	jfloat	float
GetDoubleField( )	jdouble	double

使用 GetObjectField 函数是因为所要访问的目标字段是一个 Java 对象。同样,我们也必须声明一个相应的本地类型,用来在本地方法中存储所要访问的 Java 字段。

最后,我们看到,可以通过使用适当的 SetField( ) 函数改变对象字段的值。使用 SetField( )函数是因为所要设置的目标字段具有 int 类型。与前面一样,声明正确的本地类型以存储传送到 Java 环境的本地字段的值也非常重要。

表 49.3 给出了各种可以使用的 SetField( )函数。

表 49.3 设置 Java 字段值的访问器函数

SetField 例程名	本地类型	Java 类型
SetObjectField( )	jobject	Object
SetBooleanField( )	jboolean	boolean
SetByteField( )	jbyte	byte
SetCharField( )	jchar	char
SetShortField( )	jshort	short
SetIntField( )	jint	int
SetLongField( )	jlong	long
SetFloatField( )	jfloat	float
SetDoubleField( )	jdouble	double

表 49.4 给出了本地函数中可以使用的各种原始类型。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 49.4　原始类型

Java 类型	本地类型	描述
boolean	jboolean	无符号 8 位
byte	jbyte	有符号 8 位
char	jchar	无符号 16 位
short	jshort	有符号 16 位
int	jint	有符号 32 位
long	jlong	有符号 64 位
float	jfloat	32 位
double	jdouble	64 位
void	void	N/A

49.6　从本地方法访问 Java 方法

从一个本地方法中激活一个 Java 方法的过程有些复杂。然而，使用程序清单 49.6 和程序清单 49.7 中所演示的技术，我们也可以从一个本地方法中调用其它的本地方法。事实上，如果需要的话，一个本地方法甚至可以递归调用其自己。程序清单 49.6 就是从一个本地方法使用传递给它的参数调用一个 Java 方法的很好的例子。程序清单 49.7 给出了从本地方法访问 Java 方法的 C 语言实现及实际 JNI 语法说明。

程序清单 49.6　SayHello3.java——演示从本地方法访问 Java 方法的 Java 类

```
public class SayHello3 {

        public  String rectArea(int l, int b ) {
            return "The area of the rectangle is " + l*b + " units" ;
        }
        public native void calc();

        static {
            System.loadLibrary("sayhello3");
        }

        public static void main(String[ ] args) {
            new SayHello3().calc();
        }
}
```

程序清单 49.7　SayHello3.c——演示 Java 方法访问的本地方法实现

```
# include "SayHello3.h"
# include < studio.h >
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
JNIEXPORT # void JNICALL # Java _ SayHello3 _ greet ( JNIEnv * env, jobject this) {
    jmethodID jm;
    jclass jc;
    jobject jo;
    const jbyte * area;
    jint x,y ;
    x = 20;
    y = 20;
    jc = ( * env) - > GetObjectClass(env, this);
    jm = ( * env) - > GetMethodID(env,jc,"rectArea","(II)Ljava/lang/String;",x,y);
    jo = ( * env) - > CallObjectMethod(env,this,jm,x,y);
    area = ( * env) - > GetStringUTFChars(env,jo,0);
    printf("%s \ n",area);

}
```

Java 类文件和共享库与前面相同,执行程序时其输出结果为:

The Area of the Rectangle is 400 units

从本地方法调用对象方法时应遵照以下五个步骤:

1. 获得调用对象的类类型。
2. 获得方法 ID。
3. 初始化需要作为参数传送给 Java 方法的变量。
4. 使用适当的 CallMethod( )访问器函数调用方法。
5. 根据需要转换返回的对象字段以便在本地方法中使用。

查看表 49.1 中的类型签名,可以推断出程序清单 49.6 中的 Java 方法 rectArea()的类型签名为(II)Ljava/lang/String,这是因为该方法使用了两个 int 参数,返回一个 String 对象。

表 49.5 给出了各种可以使用的 CallMethod( )访问器函数。

表 49.5 从本地方法调用 Java 方法的访问器函数

CallMethod 例程名	本地类型	Java 类型
CallVoidMethod( )	void	void
CallObjectMethod( )	jobject	Object
CallBooleanMethod( )	jboolean	boolean
CallByteMethod( )	jbyte	byte
CallCharMethod( )	jchar	char
CallShortMethod( )	jshort	short
CallIntMethod( )	jint	int
CallLongMethod( )	jlong	long
CallFloatMethod( )	jfloat	float
CallDoubleMethod( )	jdouble	double

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

49.7　访问静态字段

到现在为止，我们已经知道了如何使用访问器函数访问实例字段和方法。我们也可以使用下面将介绍的一些特殊函数，以非常类似的方法访问 Java 对象的静态字段和静态方法。

GetStaticFieldID( )：可以从表 49.1 中获得所访问字段的签名。当获得字段签名后，getStaticFieldID( )的使用与 getFieldID( )函数的使用完全一样。

GetStaticField( )：表 49.6 中给出了各种可以用来访问静态字段的 GetStaticField( )函数。选择什么样的函数要根据所访问的 Java 字段的数据类型而定。

表 49.6　访问静态字段的访问器函数

GetStaticField 例程名	本地类型	Java 类型
GetStaticObjectField( )	jobject	Object
GetStaticBooleanField( )	jboolean	boolean
GetStaticByteField( )	jbyte	byte
GetStaticCharField( )	jchar	char
GetStaticShortField( )	jshort	short
GetStaticIntField( )	jint	int
GetStaticLongField( )	jlong	long
GetStaticFloatField( )	jfloat	float
GetStaticDoubleField( )	jdouble	double

SetStaticField( )：表 49.7 给出了所有可以用于修改静态 Java 字段值的访问器函数。选择使用函数时要依照目标静态字段的类型而定。

表 49.7　设置静态字段值的访问器函数

SetStaticField 例程名	本地类型	Java 类型
SetStaticObjectField( )	jobject	Object
SetStaticBooleanField( )	jboolean	boolean
SetStaticByteField( )	jbyte	byte
SetStaticCharField( )	jchar	char
SetStaticShortField( )	jshort	short
SetStaticIntField( )	jint	int
SetStaticLongField( )	jlong	long
SetStaticFloatField( )	jfloat	float
SetStaticDoubleField( )	jdouble	double

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

49.8 访问静态方法

静态 Java 方法可以像实例方法一样激活，但所使用的函数为 GetStaticMethodID( )和 CallStaticMethod( )。

GetStaticMethodID( )：该函数的用法与 GetMethodID( )的用法相同，在激活函数之前，开发人员必须从表 49.1 中确定方法的签名。

CallStaticMethod( )：表 49.8 给出了可以使用的 CallStaticMethod( )访问器函数列表。开发人员必须根据函数调用所返回数据的 Java 类型，选择适当的函数。

表 49.8 调用静态 Java 方法的访问器函数

CallStaticMethod 例程名	本地类型	Java 类型
CallStaticVoidMethod( )	void	void
CallStaticObjectMethod( )	jobject	Object
CallStaticBooleanMethod( )	jboolean	boolean
CallStaticByteMethod( )	jbyte	byte
CallStaticCharMethod( )	jchar	char
CallStaticShortMethod( )	jshort	short
CallStaticIntMethod( )	jint	int
CallStaticLongMethod( )	jlong	long
CallStaticFloatMethod( )	jfloat	float

49.9 本地方法的异常处理

JNI 方法可以抛出异常，该异常随后可在所调用的 Java 对象中得到处理。下面是在 JNI 接口指针中可以使用的用于处理异常的一些重要函数：

Throw

```
jint Throw(JINEnv * env,jthrowable obj);
```

该调用抛出一个 java.lang.Throwable 对象。

参数：

env：JNI 接口指针

obj：一个 java.lang.Throwable 对象

返回：

0：成功

负数：失败

ThrowNew

```
jint ThrowNew(JNIEnv * env,jclass classz, const char * msg);
```

该调用使用诊断信息 msg 初始化并构造一个新的异常对象实例，类型为 classz。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

参数：

env：JNI 接口指针

classz：java.lang.Throwable 的一个子类

msg：用于类构造器的一个诊断信息

返回：

0：成功

负数：失败

FatalError

```
void FatalError(JNIEnv * env, char * msg);
```

该调用发出一个不可恢复的致命错误信息。

参数：

env：JNI 接口指针

msg：错误信息

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 50 章　Java 与 C(++)

本章内容：

50.1　共同的祖先
50.2　基本 Java 语法
50.3　Java 程序结构
50.4　Java 数据类型
50.5　对象和类
50.6　集合类型：字符串、数组及向量
50.7　类的层次与继承
50.8　语句
50.9　名字空间

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

50.1　共同的祖先

Java 和 C++ 语言都有着一个共同的祖先即 C 语言，所以，这三种语言有着许多共同的特性和语法结构。但是很显然，任何 C 或 C++ 程序员在阅读本书时都会发现它们也有许多不同之处。这些语言很相似但却并不完全相同。C 程序员将发现 Java 的大部分内容都是新添加的内容，它形成了 C 的一个超集。确实也有一些让人欣赏(或痛恨)的 C 语言特性在 Java 中不存在了——最显著的一点就是对预处理器和指针的支持。但是在更多的其它方面，Java 引入了新的结构和新的特性。最明显的就是 Java 对面向对象的编程模式的支持。

对 C++ 程序员来说，他们会发现 C++ 与 Java 的差别要小一些，有些甚至相当细微。例如 for 循环控制语句中所声明的变量的范围。不幸的是有些方面的差别则相当大，例如 Java 对象引用模型和对多态性行为的支持等。

本章对 C、C++ 和 Java 进行简要的比较。如果我们以前有 C 或 C++ 的编程经验，则可以利用本章的内容提高对 Java 的认识和使用水平。如果我们没有以前的经验，则可以通过本章的学习了解一下 Java 的发展过程。不管我们有什么样的背景，都可以将本章内容作为一种快速参考资料。

50.2　基本 Java 语法

C/C++ 与 Java 之间存在着许多细小的差别。这些差别没有什么难以让人理解的，但其中也有少数几点人们通常很难注意到。在本节中我们首先介绍它们之间最基本的一些差别。

50.2.1　词汇结构

Java、C 及 C++ 的词汇结构基本上是一样的。程序的源代码被分解成一个个标识，这些标识结合起来形成表达式和语句。所有这三种语言都是自由格式的语言，这意思是说在源文件流中各个标识的位置没有什么特别的规则。空白字符(包括空格、制表符、换行符等)可以用在任何标识的结合处用以分隔各个标识。与在 C 和 C++ 中一样，分号(;)用来结束语句。任何要求使用单条语句的地方，都可以用括号括起多条语句，形成一个独立的复合语句。

警告：虽然空白字符可以用于分隔标识，但它不能用来分隔一个标识中的各个部分。例如，空白字符不能放在一个操作符如 ++ 或 * = 的两个字符之间。

50.2.2　注释

在 Java 中有三种形式的注释：

- 传统的 C 注释，以/* 引导，然后是所有注释文本的内容直到遇到第一个 */。
- BCPL 注释，以//引导，然后继续到当前行的结束处，当前行上所有//右边的内容为注释。
- Java 文档注释，以/** 引导，然后是所有注释文本内容，直到遇到第一个 */。虽然 Java 文档注释看上去与传统的 C 注释一样，但它有着特殊的含义。

参见第四章“JDK 工具：Javac、AppletViewer、Javadoc”。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下面的代码段说明了三种风格的注释：

```
/* 这是一个多行注释的第一行
这是这一注释的第二行 */
int tableSize;//这是一个单行注释
/** 这是一个 Java 文档注释 */
```

> **注意**：与在 C 和 C++ 中一样，Java 注释不允许嵌套，请看下面的例子：
>
> ```
> /* 这是一个注释的一行
> /* 有的人可能认为这是一个嵌套注释 */
> 但这一行已不再是注释的一部分了
> */
> ```

警告：许多 C 和 C++ 程序员习惯于能够进行嵌套注释，尽管存在上面的注意中所提到的问题。这是因为有些 C 和 C++ 编译器开发商对他们的产品增加了支持嵌套注释的功能。不要被这种情况所蒙骗，以为这些扩充是 C 或 C++ 语言的一部分——其实并不是。

50.2.3 丢弃的内容

Java 对 C 和 C++ 的有些结构不再提供和支持。其中的一些如缺少预处理器我们将在本章稍后给予特别的介绍。其它不再支持部分的差别要细小一些，我们集中在这里给予说明。在大部分情况下，对于这些不再支持的性能使用其它的 Java 语法可以取得相似的结果。

无指针 Java 中不支持指针，虽然 Java 的引用模型与 C++ 中指针的使用非常相似。Java 中没有指针操作符，也不存在通过地址直接操作内存的手段。另外，内存的分配与释放也不能像在 C 和 C++ 中那样，根据程序员的需要进行分配或释放。

无 C++ 引用 在 C++ 中，可以定义引用变量来引用任何本地或抽象数据类型。在 Java 中，所有类对象类型都是通过引用操作的，所以再无需特殊的语法。然而，在 Java 中不存在声明一个引用为一个本地类型的方法。相反，使用了定义在 java.lang 中的一个 Java 包装类（例如，Integer、Float 等）。

参见 50.5.4“原始数据类型和 java.lang 封装类”一节。

无结构（Struct）、联合（Union）及枚举（Enum）类型 C 和 C++ 严密的结构不再存在。C 的结构可以通过 Java 中定义一个所有数据成员为公共的并且没有方法的类来模仿。枚举也不再得到支持，但可以通过定义一个没有方法并且所有数据成员均为公共、静态、最终成员的类来模仿。C 和 C++ 的联合也不再得到支持，并且也存在可以提供类似功能的手段。

无 const 虽然 Java 也保留了 const 关键字，但目前已不再使用。在 C++ 中，鼓励程序员使用 const 修饰符，以减少用 # define 定义常量。在 Java 中使用 final 修饰符可获得类似的效果：一个类中声明为最终成员的数据成员必须在声明时进行初始化，并且在这之后不能再对其进行修改。

50.2.4 运行时库

C 程序员习惯于在他们所编写的每一个程序中使用 C 运行时库。C 程序员最初所建立的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第一个程序一般是传统的"Hello World"程序,其中有下面的代码行:

```
printf("Hello, World! \n");
```

printf 函数是在 C 运行时库中定义的,可以在任何 C 程序中使用。

在 C++ 中,对应的功能通常由 iostream 类库中的类和方法提供。C++ 程序员最初的传统程序一般包含如下所示的语句:

```
count << "Hello again, World! \n";
```

由 iostream 类库所提供的功能同样可用于任何 C++ 程序。

在 Java 中,我们所希望的 C 运行时库或标准 C++ 类的功能是在 Java API 中提供的。API 定义了大量的类,每个类中都有许多不同的方法。传统情况下的第一个程序可能包含的语句为:

```
System.out.println("Hello from Java!");
```

println 方法定义在 PrintStream 类中,System.out 是 PrintStream 类的一个实例。最重要的是该功能由 Java API 提供,而 Java 包含了 C 的运行时库及更多的功能,它所提供的功能可用于我们所编写的每一个 Java 程序中。

50.3　Java 程序结构

当我们第一次读 Java 代码时,会对两种东西产生深刻印象。第一个是正如前一节中所列出的 Java 语法的微小差别;另一个是 Java 程序的整体结构。

50.3.1　程序总体框架

在 C 中,一个程序的总体结构是多个可相互交互的过程或函数的集合。程序以一个称为 main( )的函数为开始,然后是所有函数的一一调用。在 Java 中程序的总体结构为类定义及相互关联和交互的实例的集合。从某种程度上说这就是面向对象编程的基础。

Java 程序本身就是一些类,并且可分为两类。应用程序是含有一个 main 方法的一些类,而 Java 小程序是 Applet 类的一些扩展类。

> **注意**:应用程序与 Java 小程序的定义并非相互孤立的。通过创建一个类,该类既是 Applet 类的扩展,又有一个 main 方法,则可以编写一个既可以作为应用程序又可作为 Java 小程序使用的程序。

从这一方面及其它许多方面看,C++ 是介于 C 和 Java 之间的语言。C 程序是一些函数的集合;Java 程序是一些类的集合;而 C++ 程序则是一些函数和类的集合。

50.3.2　只有方法没有函数

正如所指出的,C 程序只是一些函数的集合。在 C++ 中,大部分的程序是一些类的集合,类似于 Java 程序。然而,C++ 程序使用一个称为 main 的函数开始执行程序,就像 C 程序一样。另外,C++ 除了作为类定义一部分的方法之外,还提供了对于其行为与 C 函数类似的函

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

数的明显支持。Java 在其面向对象方面要比 C++ 纯洁得多，只提供了对类中所定义的方法的支持，而不提供对函数的支持。

提示：如果我们必须创建功能类似于 C 和 C++ 函数（例如，数学库）功能的代码，可通过创建一个只是由静态方法和静态数据项组成的工具类来实现。

50.3.3 无预处理器

当许多 C 和 C++ 程序员知道 Java 中不再有预处理器时，他们对此感到很疑惑。事实也确实如此，Java 不再支持预处理器，因为语言本身不再需要它。

常量值 在 Java 中没有使用 # define 定义的常量。正如前面所看到的，数据元素在类定义中可以声明为最终型的，这样它们的初始化值就是再也不能改变的了。为了得到最接近于 # define 的功能，我们可以在建立这些数据元素时将其声明为公共、静态和最终型的成员。

注意：Java 中与 C 的 # define 等价的功能更强大，它的名字受到编译器的保护，它具有真实的数据类型，它的使用可以得到编译器的正确性检查。

宏 C 语言中宏也是使用预处理器中的 # define 结构实现的，其设计目的是为了提供一种可以在编译阶段实现的类似于函数的结构，以避免运行时与函数调用有关的开销。Java 没有提供与 C 宏等价的功能。但必须记住，C 的宏是极易出错的，并且经常是程序故障的根源。而在 C++ 中，鼓励程序员放弃使用 # define 定义宏，而使用 C++ 提供的内联函数。Java 没有显式地支持内联（inline）关键字，但是 Java 编译器可以自由地内联任何函数，并将其作为一般优化处理的一部分。

源文件包含 Java 不提供任何等价于 C 的用于源文件包含的 # include 指令的功能。实际上在 Java 代码中这是根本不需要的。C 语言中 # include 的两个最常见用途是用来定义 # define 常量和声明函数原型。正如已经看到的，Java 采用了完全不同的另外一种语法来支持常量值，这种语法不需要使用源文件的包含。另外，由于 Java 不支持脱离于其定义的方法声明，所以也不需要使用原型声明。

注意：C 编译器只支持很有限的向前引用形式，即标识——这里是指函数名——必须在使用它之前给出定义。实际中，这就是说 C 程序员必须或者提供一个函数原型（即一个声明），或者在第一次使用函数之前保证给出函数的定义。而 Java 中则支持非常丰富的向前引用，所以也就不再需要单独的函数声明。

下面的代码段说明了刚刚讨论过的不同的预处理器问题，第一段代码为 C++ 代码：

```
# include < iostream.h >
# define PLAYERS 2
Class Game
{
public:
Game( )
        {cout < < "Constructing the Game \ n";}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
};
```

现在看看使用 Java 的等价代码：

```
import java.lang.*;
Class Game{
    public Game( )
    {System.out.println("Constructing the Game");}
    public static final int PLAYERS = 2;
}
```

注意 import 语句是如何替换 #include 语句的。这两个语句看上去非常相似，但实际上所完成的任务完全不同。#include 语句以 C 预处理器的角色进行了一个直接源文件包含。由于预处理器不能识别 C 语法(它基本上是一个批处理模式的文本编辑器)，所以所包含的文件中可以包含任何内容。import 语句则不实际包含任何东西，它只告诉编译器到何处去寻找所需要使用的类。

> **提示**：在上面的例子中导入了 java.lang 软件包。事实上这个包也是 Java 编译器要自动导入的一个包，不管有没有 import 语句。不显式导入 java.lang 包也可以使用其中的类。

50.3.4　源文件名

目前，大多数 Java 开发工具包括 Sun 的 JDK，都要求将一个公共类的定义包含在一个与之同名的源文件中。例如，InventoryItem 类的定义必须包含在一个名为 InventoryItem.java 的源文件中。

虽然目前的命名限制仅作用于公共类的定义，但 Java 的使用人员普遍习惯于在每个源文件中只定义一个类，不管所定义的类是公共的还是非公共的。这种实际中所使用的方式对于许多 C 和 C++ 程序员来说，可能有些陌生，他们可能会对创建一个简单的 Java 程序而使用到的源文件个数感到惊讶。但这种做法的确使得 Java 源代码可以很容易地找到——不管是对我们自己还是对编译器来说——并且可以保持文件命名时某种程度上的一致性。

50.4　Java 数据类型

Java 支持比 C 和 C++ 更强大更特殊的数据类型，本节中我们将讨论整型数据类型、统一码(Unicde)字符、布尔型、浮点类型及集合类型。我们还将看到从 Java 的一种数据类型到另一种类型的转换。

50.4.1　整型数据类型

正是在整型数据类型中我们看到了 Java 和 C/C++ 之间的大部分区别。在 C/C++ 中大多数依赖于平台的部分 Java 都给出了清晰的定义。表 50.1 总结了这些区别。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 50.1 Java 和 C/C++ 数据类型比较

数据类型	C/C++	Java
byte	不存在，由 char 代替	通常使用 8 位
char	8 位，一个 ASCII 或 EBCDIC 字符	16 位，一个统一码字符
short	至少 16 位	16 位
int	通常为 16 位或 32 位	32 位
long	至少 32 位	64 位

注意：在 Java 中没有无符号整型数据类型。

50.4.2 统一码字符

在 Java 中，char 和 String 数据类型不是单字节的 ASCII 或 EBCDIC 值，而是统一码字符集中的 16 位值。更大的数据类型意味着可以支持更多可能的值，给非英语或多语言环境下的程序员提供更大的灵活性。在统一码集合中，前 256 个字符是与 ASCII 字符 0x00 - 0xFF(这也是 ISO 标准 ISO8859 - 1)一致的。

统一码字符集中的特殊值在 Java 中可以使用/uhhhh 语法来表示，其中 h 是一个十六进制数字，表示统一码字符 16 位中的四位。事实上，我们无需指定全部四位十六进制数字，如果所指定的数字位数少于 4 位，则高位设置为零。

注意：Java 也支持 C 和 C++ 中的转义序列例如 \n、\r 及 \t 等，用来表示非打印控制字符。

50.4.3 布尔数据类型

C 和 C++ 都不提供本地布尔数据类型。在这两种语言中，大多数程序员都使用 int 型变量来存放布尔值，并且习惯上用 0 值表示 false(假)，1 值表示 true(真)。另外，C 中的逻辑运算符和关系运算符均产生整型值，循环控制语句或 if 语句也都使用整型值作为其控制表达式。

在 Java 中，所有这些情况下所使用的都是本地布尔数据类型。布尔数据类型只支持两个值 true(真)或 false(假)。关系运算符和逻辑运算符均产生 boolean 类型的值，逻辑操作符只接收 boolean 类型的操作对象。Java 中的循环控制语句和 if 语句只能使用 boolean 类型的控制表达式。

警告：true(真)和 false(假)是 boolean 类型的值，正如 1、2 或 3 是整型类型的值一样。虽然这两个值在 Java 语言语法中没有被定义成关键，但它们无疑是被保留的，所以我们不能使用它们作为变量名或类名。

50.4.4 浮点类型

与 C 和 C++ 一样，Java 支持浮点和双精度数据类型。然而，在 C 和 C++ 中，这些数据类型是平台有关的，在 Java 中，所有浮点操作——包括浮点和双精度变量的取值范围——都是

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

由 IEEE 745 浮点运算标准所定义的。

> **注意**：大多数现代的 C 和 C++ 编译器也提供了与 IEEE 745 标准一致的支持。所以，对大多数程序员来说，Java 使他们已经熟悉的行为正式化了。

50.4.5　集合数据类型

集合数据类型包括字符串和数组，它们在 Java 中的实现略有不同。最显著的一点是，Java 中的 String 是一种本地数据类型，“+”操作符可以用来连接字符串。另外，数组和字符串都具有运行时完全可靠的已知长度。与在 C 和 C++ 中一样，数组必须静态地给出大小。Java 也支持动态分配大小的集合数据类型。它是以 Vector(向量)类的方式来支持的。(参见本章后面的“集合类型：字符串、数组及向量”一节。)

50.4.6　类型转换与强制类型转换

C 和 C++ 在各种场合包括表达式计算、赋值、函数调用等都支持特殊的类型转换。Java 也支持特殊的类型转换，但只能在非常严格控制的条件下进行。特别是在一个变窄转换(转换的结果使精度降低)的情况下，转换只允许以显式的强制方式进行。考虑下面的例子：

```
byte b = 126;
int i = 17;
float f = (float)3.50;
b = (byte)(b+i);  // 必须强制转换为 byte 类型
i = (int) f;  //必须强制转换为 int 类型
f = i;  //不需要强制转换，因为这不是一个变窄转换
```

50.5　对象和类

Java 和 C++ 都支持面向对象的编程方法，所以它们都支持对象和类，而 C 语言则不是这样。当我们比较 Java 和 C 语言时，所有支持面向对象的语法，包括对对象和类、对继承和接口的支持等，都是新概念。如果我们是一个 C 程序员，不熟悉面向对象的技术，最好回头参考一下第 5 章“面向对象的程序设计”和第 11 章“类”的内容。

虽然 C++ 和 Java 都支持面向对象的技术、都包含对对象和类的支持，但二者在这些方面也存在着微妙的和重要的差别。这一节我们将集中介绍二者在这方面的主要差别。

50.5.1　声明引用类型

除了原始数据类型(char、int 等等)之外，还有对象、数组、向量以及字符串等，所有这些都是通过引用而不是通过值来操作的，并且称之为引用类型。考虑如下的声明：

```
GameTable chessBoard;
```

如果 GameTable 是我们在前面某处已经定义的一个类，则上述声明在 C++ 和 Java 中都将是语法上正确的。然而，在 C++ 中，这将创建了 GameTable 类的一个新实例。在 Java 中，上述声明仅仅是变量 chessBoard 所具有的引用类型的一个声明，它是对 GameTable 的一个引

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用,但并不指向这个类的任何特殊实例。在 Java 中,我们必须使用 new 操作符实例化该对象:

```
GameTable chessBoard;
chessBoard = new GameTabe( );
```

注意:与 C 和 C++ 时一样,在 Java 中,一个变量的声明和其初始化可以合并在一个语句中:

```
GameTable chessBoard = new GameTabe( );
```

上面的这一代码行声明了变量 chessBoard 是对 GameTable 的一个引用。同时实例化了一个新的 GameTable,chessBoard 现在指向这个新的 GameTable 实例。

50.5.2 操作引用

在 Java 中操作一个对象时也必须记住,我们是在操作引用而不是实例。在下面的代码段中,变量 b1 和 b2 是截然不同的变量,所以它们可能拥有截然不同的引用,但是在赋值以后,它们都指向了同一个实例:

```
Box b1,b2;  //对类型 Box 的两个引用
b1 = new Box(2,2);  //实例化 b1
b2 = b1;  //现在 b1 和 b2 都指向
          //同一个实例
```

当我们比较引用时,Java 中的对象与引用的关系也很重要,请看下面的代码段:

```
Box b1,b2;  //对类型 Box 的两个引用
b1 = new Box(2,2);  //实例化 b1
b2 = new Box(2,2);  //以相同的值实例化 b2
if(b1 = = b2)  //这个表达式比较的结果为假
```

在这个例子的最后一行中,对引用 b1 和引用 b2 进行了比较,缺省情况下,只有在两个对象实际指向同一个实例时,这种比较的结果才会是真。由于 b1 和 b2 指向两个不同的实例(虽然具有相同的值),比较的结果必然产生一个 false(假)。

提示:Java 中所有的类都是从 Object 基类继承而来,在基类中使用上面提到的行为定义了一个 isEquals 方法。如果需要一个比较例如上面我们所看到的例子有不同的行为,只要在自己的类中覆盖 isEquals 方法并根据自己的标准产生 true(真)或 false(假)值即可。

50.5.3 方法调用:通过值调用和通过引用调用

与 C 和 C++ 一样,在 Java 中缺省的方法调用是通过值调用。然而必须记住,对对象的操作总是通过引用来进行的。这就是说当我们调用一个方法并且试图传送一个作为参数的对象时,我们实际上传送的是对该对象的一个引用,所传送的"值"是对一个对象的引用。从效果上讲,使用参数传送本地类型到一个 Java 方法时称为通过值调用,当传送引用类型时称为通过

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

引用调用。

提示:如果我们需要调用一个方法并且传送一个值的引用,则通过使用第一个对象的一个本地拷贝可以取得同样的效果,如下列代码所示:

```
Box b1 = new Box(2,2);
valueMethod(new Box(b1));
```

此处,传送到 valueMethod 方法的引用是一个使用 b1 初始化的新实例,在该方法内对该引用所做的任何修改都不会影响到 b1。

50.5.4　原始数据类型及 java.lang 的包装类

java.lang 包中定义了几个将原始数据类型封装到引用对象中的类。例如,int 值可封装到 Integer 对象中,对每一种原始数据类型都存在这样的一个封装类,这些封装类有许多有趣的使用方式,可使我们的编程更加轻松。

通过引用传送原始数据类型　在 Java 中不存在与 C++ 中引用语法类似的显式引用语法。例如,在 C++ 中一个方法可以定义为使用一个被传递参数的引用,而不是使用该参数的值。在 Java 中要通过引用传送一个原始数据类型,必须首先按照如下方式将值封装在一个对象中:

```
int j = 17;
Integer arg = new Interger(j);
methodCalledByReference(arg);
```

警告:注意,Java.lang 中的封装类不存在设置或更新它们所包含的原始数据类型值的方法,这限制了它们在某些情况下的使用,例如在前面所提到的例子中。如果我们要经常这样做,那么可考虑使用 C++ 而不是 Java。

添加原始数据类型到一个向量　Java 的 Vector 类允许我们创建动态尺寸的集合引用类型。然而,由于向量中的元素必须是引用,所以我们不能向一个向量中添加原始类型的值,而是必须将原始类型值封装在一个封装对象,然后再将新对象添加到向量。下面是一个例子:

```
Vector sizes = new Vector( );
Integer small = new Integer(5);
sizes.addElement(small);
```

int 型的值 5 被封装在了 small 对象中,该对象被添加到了 sizes 向量中。

注意:由于 Java 中的散列表也是为存储对象引用而设计,上述说明同样适用于散列表。

转换原始类型为字符串　java.lang 中的每个封装类都定义了一个称为 toString 的方法。该方法可以把原始类型的数据转换为 Java 的 String 对象。当我们只有原始数据类型的值,而需要一个 String 对象时,可使用该方法。在下面的例子中,我们需要一个浮点值的 String 形式的表示,以便可以在一个 Applet 的 paint 方法中将其作为一个参数传送到 drawString 方法:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Float temp = new Float(this.interestRate);
g.drawString(temp.toString( ),10,10);
```

正如所看到的,我们首先将 interestRate 值封装在一个 Float 对象中,然后调用 Float 的 toString( )方法创建了一个合法的 String,最后将该 String 传送给 drawString 方法。

50.5.5 对象的生存周期

与其它的数据元素或数据结构一样,Java 对象也有其明确的生存周期。对象一般要经历创建、有用的生存期、然后销毁这样的一个过程。对于作为 C++ 程序的一个部分所创建的对象也同样如此。然而,两种语言管理对象生存周期的方式却存在着差异。

对象创建时 C++ 中可以使用两种方式创建对象。如果一个对象存储在变量中,那么当该变量进入其有效范围时,对象被实例化,即创建。当该变量退出其有效范围时,该对象被销毁。然而,如果通过指针操作一个 C++ 对象,则只有在使用 new 操作符时,对象才被实例化。

Java 中的对象生存周期要比 C++ 简单一些。只有一种创建和使用 Java 对象的方式,这就是使用引用。所以,只有在使用 new 操作符创建了对象后,对象才存在:

```
Table r;  //Table 对象尚未存在
r = new Table( )  //只有到此时 Table 对象才存在
```

这与 C++ 中使用指针操作对象时的语法类似:

```
Table * r;  //创建指针的 C++ 代码
r = new Table( );  //实例化 Table 对象的 C++ 代码
```

与在 C++ 中一样,只有在对象被真正实例化时才调用构造器方法。

> **注意**:在 Java 和 C++ 中,都可以重载构造器方法。也就是说,只要构造器签名(参数的个数和类型)不同,可以同时存在多个构造器方法。

对象销毁时 C++ 中一个对象的销毁方式取决于对象的创建方式。如果一个对象存储在一个变量中,那么当该变量退出其有效范围时该对象即被销毁。如果一个 C++ 对象是使用 new 操作符创建的,那么只有在使用 delete 操作符显式地请求删除时,该对象才被销毁。

同样,Java 中的对象销毁也有略微的区别。一个对象只有当使用 new 操作符实例化时才被创建。原则上讲,只有当不存在任何对一个对象的引用时,该对象才被销毁,考虑如下的代码段:

```
Square s1 = new Square(5);
Square s2 = new Square(10);
s2 = s1;
```

当执行完最后一条赋值语句后,s1 和 s2 均指向同一个对象。对于使用 Square(10)所创建的 Square 对象已经不存在任何引用,原则上讲,也就是在此时 Square(10)所创建的对象被销毁。

为什么我们要说"原则上讲"呢?这是因为 Java 使用了一个动态内存管理即垃圾收集的形式。这就是说内存的再分配是由 Java 垃圾收集器自动完成的,并不是由程序来控制的。由于垃圾收集器通常以一个独立线程方式运行,所以,称一个对象"可以作为垃圾收集"要比说对

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间:可以联系群主获取更多大型企业内部技术教程。

象被"销毁"更准确一些。这种区别只是理论上的，不管对象是被销毁，还是仅仅准备好作为垃圾收集，实际上是该对象再也不能在程序中使用了。从实用目的来讲，该对象已经结束其使命并准备退出。

> **注意**：虽然 Java 支持构造器，但不支持像 C++ 中那样的销毁器。但是 Java 中有一个 finalize 方法，我们可以在自己的定义中覆盖该方法，当对象作为垃圾被收集时，激活该方法。由于垃圾的收集时间是不可预测的，所以 finalize 方法的使用并不像 C++ 中销毁器的使用那样普遍。

50.5.6　Java 引用与 C++ 指针

到这一节我们可能已经清楚了，Java 中操作对象的大部分语法与 C++ 中的情况非常相似——直接使用 new 操作符实例化一个对象。但是在 C++ 中使用 new 操作符时，返回给我们的是对象的一个地址，我们可以随后将该地址指定给一个指针变量。在 Java 中，new 操作符返回给我们的是一个对象引用，我们可以将其指定给一个引用变量。

有些读者特别是那些刚刚开始学习 Java 的人，可能会认为 Java 中引用的使用实际上与 C++ 中指针的使用完全一样——只是具有不同名称的同一种数据结构。这种说法有一部分是正确的，并且如果读者对 C++ 语法熟悉，会发现 Java 的语法很容易掌握。但是 Java 的引用与 C++ 指针之间存在着重要的差别。在 Java 中，所有内存的管理都是自动的，并且从不允许我们进行直接内存地址操作。Java 解释器到底是否在其内部使用了指针来实现引用，这实际上并不重要。重要的是在这一点上 Java 程序比 C 和 C++ 程序更加稳定和可靠了。

> **注意**：C++ 程序员应该对 this 是熟悉的，它是一个指针变量，包含了引起方法调用的对象的地址。Java 中也有 this，只是它是一个引用而不是一个指针。除了这一点之外，它们具有相同的含义。

50.6　集合类型：字符串、数组及向量

与 C 和 C++ 一样，Java 也提供了一些用来创建集合类型值的机制，不论这些值具有原始数据类型还是引用类型。在这一节中，我们将学习 Java 中三种最常见的集合类型：字符串、数组及向量。

50.6.1　字符串

在 Java 中，字符串的处理与 C++ 中非常相似。最主要的区别是在 Java 中当一个 String 被声明之后，其值是不能修改的。为了实际修改一个 String，必须创建一个使用 String 初始化的 StringBuffer，然后再进行修改。在这之后我们可以使用 StringBuffer 的内容创建一个新的 String。下面的代码段说明了这一过程：

```
//创建初始的 String 对象
String badString = new String("This is a Strng");
//创建我们可以修改的 StringBuffer
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
StringBuffer correction = new StringBuffer(badString);
//对 StringBuffer 对象进行修改
correction.insert(12,'i');
//使用修改过的内容创建一个新的 String 对象
String goodString = new(correction);
```

注意:与许多其它的 Java 对象一样,Java 的 String 对象也有已知并且可靠的尺寸。对于 String 对象,length 方法以 int 返回它的长度。在 Java 中 String 对象是不可能像 C 和 C++ 中那样发生"溢出"的。

50.6.2 数组

Java 数组与 C 和 C++ 数组非常相似。它们是同类型数据的集合(即所有元素都具有相同的数据类型),并且都具有固定的大小。然而,二者也存在一些微妙的差别。

原始类型数组 在 Java 中,数组是隐含的数组类的一个实例——隐含的意思是没有 Array 关键字来表示该类名。虽然如此,数组必须像其它对象一样进行实例化。下面的例子声明 myArray 为一个 int 型的数组:

```
int[] myArray;
```

注意:在 Java 中,说明 myArray 为一个数组引用的空方括号对可以放在数据类型的后面(如上面的例子),也可以放在变量名的后面:

```
int myArray[];
```

将方括号对放在类型名的紧后面是 Java 的惯用风格。

但此处的 myArray 是一个引用变量,它并不指向任何东西。我们还必须实例化该数组对象:

```
myArray = new int[10];
```

提示:Java 数组的大小在编译时才确定,与 C 和 C++ 中完全一样。如果我们需要创建一个动态大小的数组,可使用向量。

与在 C++ 中一样,上面所讲的两个步骤可以合并为一个语句,如下所示:

```
int[] myArray = new int[10];
```

C 和 C++ 程序员习惯于在声明数组时对数组进行初始化,所使用的语法格式类似于如下语句:

```
int powers[3] = {3,9,27};
```

在这种情况下数组的大小可以忽略,编译器将根据所提供的初始值的个数,确定数组的大小。

在 Java 中也可以使用同样的语法,下面的例子完成了三项明确的任务:它声明 powers 为

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

一个 int 型的数组;实例化该数组;初始化数组中的每个元素。数组的大小由初始化值的个数所决定:

```
int[ ] powers = {3,9,27};
```

当数组作为参数被传送到一个方法时,同样使用空的方括号对来说明一个数组。下面的例子给出了一个 int 型的数组及一个 char 型的数组,这两个数组均在一个方法的定义中作为参数被接受:

```
public syntaxExample(int[ ] thisInt,
    char[ ] thisChar)
{
//此处为方法体
}
```

引用数组　与 C 和 C++一样,Java 中也可以有非原始类型的数组,这种数组为引用类型的数组,即这种数组是一个由引用组成的数组。最基本的语法仍然成立。下面的例子创建了一个 Month 引用数组:

```
Month[ year = new Month[12];
```

当执行完上述代码行之后,一个由 12 个 Month 引用组成的数组将被实例化,并且 year 将是指向该数组的一个引用变量。但该数组中的 12 个元素都还未指向任何东西,即我们已经实例化了这个数组,但未实例化 12 个 Month 中的任何一个。如果 Month 类有一个以 String 作为参数的构造器,我们可以使用如下语句继续实例化数组中的每一个元素:

```
year[0] = new Month("January");
year[1] = new Month("February");
year[2] = new Month("March");
//... 等等
```

> **注意**:Java 数组与 C 和 C++的数组一样,使用以零开始的下标。所以,一个有 12 个元素的数组所具有的有效下标为 0 到 11。

> **提示**:Java 数组与 Java 的 String 一样,有着固定和可靠的大小。每一个数组对象都有一个与之关联的 length 变量,该变量中存放着数组的准确长度。在前面的例子中 year.length 为 12。

50.6.3　向量

在 C 和 C++中,由程序员控制之下的动态内存分配是一个由来已久的传统——不幸的是这是一个多少年来已经产生了一些非常不稳定的代码的传统。在 Java 中,这种动态内存分配方式已经全然不可能了。然而,许多情况下数据结构的动态尺寸对于问题的解决会非常关键,在这种情况下 Java 程序员可使用 Vector 类。

Java 的 Vector 类与动态尺寸的数组类似,它包含有动态数量的对象引用。使用 addEle-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

ment 和 removeElement 方法，可根据需要对向量添加或删除引用。由于所有的 Java 类都是从 Object 类继承而来，所以 Java 的向量元素可以是任何 Java 类的引用。

注意：当从向量中提取一个引用时，该引用具有 Object 类型。在引用可以可靠地使用之前，必须将其强制转换成适当的引用类型。

向量中的每一个元素都是被编号的，其开始的元素被编号为 0。这与 Java 数组的下标是一致的，也与 C 和 C++ 中数组的使用一致。Java 中的向量与 Java 数组和字符串一样，具有可靠的大小。Vector 类中的 size 方法返回当前存储在向量中的元素个数。

下面的代码段给出了一个 Month 向量，它与前面我们介绍的 Month 数组非常相似：

```
MonthVector = new Vector( );
MonthVector.addElement(new Month("January"));
MonthVector.addElement(new Month("February"));
MonthVector.addElement(new Month("March"));
//等等 ...
```

有关向量更详细的信息，参阅讲述 java.util 的第 33 章。

参见 2.5 中“java.util”部分。

50.7 类的层次与继承

与大多数 C++ 程序一样，大多数的 Java 程序广泛地使用了继承，这种情况下一个类是根据另一个类而定义的。新的类也称为派生类或子类，它继承了原始类的所有特性——原始类也称为基类或超类。当一个基类本身继承了来自于另一个类的特性时，这种结果称为类层次。（关于继承更详细的信息，参见关于面向对象的程序设计的第 5 章。）

注意：在类层次中，一个类可以同时是一些类的子类和另一些类的超类，我们对此不应感到惊奇，在现实生活中，某人的父亲同时也是某人的孩子。

在 Java 中，所有的类都是 Object 基类的子类，并且继承了它的所有特性。

50.7.1 继承的语法

Java 中创建派生类的语法不同于 C++。当从一个超类中派生一个子类时，在新类的定义中 Java 使用了关键字 extend，下面的例子对 C++ 和 Java 语法进行了对比。在这两个例子中，我们都创建了一个称为 CaptionedRectangle 的类，该类从 Rectangle 基类继承而来。

首先，在 C++ 中：

```
class CaptionedRectangle:public Rectangle
{
    //此处为类的定义
}
```

下面为 Java 中对应的定义：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
class CaptionedRectangle extends Rectangle
{
    //类定义
}
```

50.7.2　instanceof 操作符

instanceof 操作符提供了不少方便。该操作符使用了两个操作对象:左边为一个引用,右边为一个类的名字。如果引用所指的对象是右边的类或其任何子类的一个实例,该操作符返回 true(真)。在下面的例子中,只有当 obj 指向 Vehicle 类的一个对象时我们才需要调用 checkInsurance 方法:

```
void quarterlyUpdates(Object obj)
{
    if(obj instanceof Vehicle)
    {
        obj.checkInsurance( );
    }
}
```

注意:由于所有类都是由 Object 派生出来的,表达式(obj instanceof Object)将永远返回 true(真)。

50.7.3　继承和多态性

在 Java 中,声明为某个类类型的变量实际上可以指向该类或其任何子类的对象。以前面的 Rectangle 和 CaptionedRectangle 类为例,我们可以有如下的代码:

```
void updateScreen(Rectangle r)
{
    r.display( );
}
```

如果 r 恰好指向一个 Rectangle 对象,则调用 Rectangle 类中的 display 方法(如果有的话)。如果指向一个 CaptionedRectangle 对象,则激活 CaptionedRectangle 类中的 display 方法。这就是多态性的基础,它是面向对象程序设计的一个概念。在这种情况下一个单个的接口(此处为 display 方法)实际上可以有多种实现。

注意:C++ 程序员对这一概念是熟悉的。然而,对于体现这一功能的 C++ 类层次,基类中的 display 方法必须声明为 virtual 型的,而 Java 中的方法在 C++ 看来全都是"virtual(虚拟)"型的。

50.7.4　接口与多重继承

在 C++ 中,一个类可继承多个基类的特性——这就是多重继承的概念。与此相反,Java

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的类只能继承一个基类的特性。但是一个 Java 类可以声明为实现(implement) 一个 Java 接口。例如,我们可以有下面的代码段:

```
class CaptionedRectangle
extends Rectangle
implements Displayable
{
//类定义
}
```

Java 接口类似于类,但其数据项全都是静态的和最终的,并且其方法全是抽象的。在一个新类的定义中,接口中的每个方法必须给出具体的定义。接口实际上是一种保证形式:如果定义一个类来实现一个接口,它是对使用该类的用户的一种保证,即接口中的每个方法在该类中都有完整的定义。关于接口我们在第 12 章"接口"中有深入讨论。

> **注意**:C++ 中的多重继承机制存在一些很微妙的问题。作为一个结果,C++ 程序中多重继承最可靠和最常见的使用是在有一个继承主链,并且任何用作基类的其它类都是实用方法和常量值的集合的时候。这种行为在 Java 的接口结构中成了正式化的东西。

50.7.5 超级引用

正如我们已经看到的,this 引用指向一个引起一个方法调用的对象,我们还有一个 super 引用,它指向 this 对象的父类。当派生类的一个方法需要显式激活超类中的一个方法时,便可以使用这种引用。考虑如下例子,此处 display 方法是 CaptionedRectangle 类中的一个方法:

```
void display( )
{
    System.out.println(this.caption);
    super.display ( );
}
```

super 引用也可以用于一个基类构造器方法的第一行,以显式地激活一个超类的构造器。另外,比较一下 C++ 和 Java 在这方面的情况也是有意义的。下面的代码段是 C++ 中可能出现的,类 CaptionedRectangle 构造器使用了五个参数:其中的四个定义了矩形 Rectangle,另外一个定义了 CaptionedRectangle 的标题:

```
//C++ 构造器
CaptionedRectangle::CaptionedRectangle
    (int x,int y,int width,int height,String caption):
Rectangle (x,y,width,height)
{
    // CaptionedRectangle 构造器体
}
```

Java 中对应的代码应为:

```
//Java 构造器
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
CaptionedRectangle
(int x,int y,int width,int height,String caption)
{
    super(x,y,width,height);
    //CaptionedRectangle 构造器的其余部分
}
```

50.7.6　没有范围解析操作符

许多 C++ 程序员对 Java 中没有范围解析操作符(双冒号::)感到吃惊。事实上,Java 中不存在可以选择的手段以激活一个类层次中某个指定的类中的方法。

提示:如果我们所要激活的方法在超类中,可以使用 super 引用激活该方法。

注意:许多新的 Java 程序员试图使用多重 super,例如 super. super. super.display( ),来激活层次中其它地方的一个方法。这倒是一个很有趣的主意,但 Java 不支持。

50.8　语句

基本上说 Java 支持与 C++ 相同的控制语句,与原来的 C 语言的语句也非常相似。本节中我们讲解它们之间差异。

参见第 10 章"控制流程"。

50.8.1　循环

Java 中的循环语句基本上与 C++ 是一样的。然而,在 Java 中可以给循环添加一个标签。该标签可以作为 break 或 continue 语句的参数使用。下面是一个例子:

```
start:
for(int j = 0;j<20;j++)
for(int k = 0;k<20;k++)
for(int l = 0;l<20;l++)
if((j+k+l) = =20)break start;
```

当执行到 if 语句中的 break 语句时,最外层的循环被中断。在 C 或 C++ 中如果不使用 goto 语句是很难取得这种效果的。

注意:由于这是实际中一般可接受的 goto 语句的唯一用法,而现在已不再需要使用 goto 语句来获得这种效果,所以在 Java 语法中取消了 goto 语句。

在前在的例子中我们已经注意到,控制变量(j、k 和 l)是在循环语句中定义的。这种情况在控制变量仅仅与循环内部有关,在循环外部没有任何意义时非常方便。这种用法在 C++ 中也是合法的,但是存在着微小的差别。在 C++ 中,控制变量的有效范围是从 for 循环控制语句开始一直到块结束。而在 Java 中,控制变量的有效范围仅仅是循环体本身,块中的其它地

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

方是没有定义的。

50.8.2 条件语句

Java 的 if 语句中的显式条件及各种 Java 循环中的隐式条件均需要能够产生一个布尔值的表达式。在 C 或 C++ 中则并非如此。在 C 或 C++ 中，表达式可以产生任何值；非零值被认为是真值，零值被认为是假值。这样就消除了完整的一类 C 和 C++ 错误。

```
//最常见的 C 错误
//幸运的是这种情况在 Java 中不再编译
if(x = 5)
{
}
```

50.8.3 同步语句

Java 特性中对多线程的支持带来一些问题。特别是当可能有多个线程同时修改对象的代码段时，可能会引起对象毁坏。synchronized 语句通过阻塞代码执行直到获得对象的独占访问这种办法，解决了这一问题。

synchronized 语句的语法为：

```
synchronized(expression)statement
```

其中，expression 产生一个待保护对象的引用，statement 是一个代码块，当获得对对象的主控权之后，执行这一代码块。

```
public swapFirstValues(int[] k)
{
synchronized(k)
{
int temp;
temp = k[0];
k[0] = k[1];
k[1] = temp;
}
}
```

警告：如果一个对象从来不会有多个线程访问它，则不要使用 synchronized 语句，因为这样会引起运行时不必要的处理开销。

关于线程更详细的介绍参见第 13 章“线程”中的有关内容。

50.8.4 操作符与表达式

Java 所支持的操作符基本上与 C++ 一样，包括操作符的处理优先级和它们之间的关系（见第 9 章）。下面是二者的主要区别：

- 我们已经看到，Java 增加了 instanceof 操作符。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

参见本章前面的 50.7.2“instanceof 操作符”一节。

- “+”操作符可用来连接两个 String 对象。
- Java 中右移操作符(>>)显式并有符号地扩展值,而新的逻辑右移操作符(>>>)使用 0 填充空白。
- 正如我们所看到的,关系运算符和逻辑运算符均产生布尔值,并且逻辑运算符只使用布尔值作为操作对象。

参见 50.4.3“布尔数据类型”一节。

- 取消了范围解析操作符。

参见 50.7.6“无范围解析操作符”一节。

- 取消了逗号操作符。

> **注意**:虽然取消了逗号操作符,但逗号仍可以使用在 for 循环控制语句中,用来分隔多个控制表达式。如下所示:
>
> ```
> for(i = 5, j = 0; i > j; i--, j++)
> ```

与在 C++ 中不同,Java 操作符不可以重载。这就是说一个操作符的含义是由语言的语法固定了的。我们所需实现的任何特殊的行为必须通过使用显式的方法调用来完成。

Java 表达式的计算采用了比 C 或 C++ 更加容易预测的方式。在任何包含一个操作符和两个操作对象的表达式中,左操作对象总是首先计算,然后才是右操作数的计算,在方法调用中,参数列表的计算严格遵守自左至右的规则。

50.9　名字空间

名字空间主要定义一个名字或符号的作用范围——即程序或类集合中的名字或符号具有意义的部分。更为重要的是,明确名字空间可以防止变量和方法名的冲突——即所谓的名字冲突。

Java 和 C 或 C++ 之间首要的也是最简单的区别是 Java 程序中不存在任何种类的全局变量和方法。这使得名字空间的不合法及冲突减少到了最低程度。

Java 还引用了包的概念,帮助我们管理自己的名字空间。一个包是类的一个集合,每个包都有其自己的名字空间。在实际使用中,这意味着一个类的名字与其相关联的包结合在一起,产生一个全局唯一的类名。由于方法名和类名是在一个类的内部进行管理的,这就基本上消除了出现名字冲突的可能性。

参见 11.10“包”一节。

> **注意**:由于 Java API 的类和方法都属于预先定义的包,所以某人有意或无意创建出与系统所提供的类或方法发生冲突的类或方法的可能性是没有的。这是从系统安全性和防止用户程序错误的角度来考虑的。

Java 的设计从底层直到 Internet。包及其所提供的名字空间保护对于在分布式 Internet 环境下提供安全性和可靠性是必要的。但是包不仅仅是由于 Internet 才变得如此必要,它对于消除 C 和 C++ 中可能出现的名字冲突也很有帮助。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第八部分　调试 Java

第 51 章　调试 Java 代码
第 52 章　理解 .class 文件
第 53 章　Java 虚拟机的内部机制

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 51 章　调试 Java 代码

本章内容：

51.1　sun.tools.debug 包的体系结构

51.2　深入探讨 JDB

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

51.1 sun.tools.debug 包的体系结构

对每个人来说，在使用像 Java 这样的新语言和新执行环境进行应用开发时所面临的问题之一是，掌握可用的适当技术和工具，以找出所编写的应用程序中存在的问题（如程序陷阱）。Java 除提供了标准的结构，用于创建设计良好的、面向对象的应用程序（继承性、封装及多态性）之外，还包括有一些新的特性如异常和多线程。这些特性使程序的调试更加复杂。

为了使 Java 尽快地进入市场，Sun 公司起初并没有打算建立一个开发环境，以支持 Java 应用程序和小程序的开发，相反，Sun 公司为像我们这样的开发人员提供了一些可使用的特性和工具，以便创建这些高级工具。这些工具之一是 sun.tools.debug 包，称为 Java Debugger (JDB) API。该 API 包含有一组类，它允许创建可以与正在 JVM 的一个本地或远程实例中运行的应用程序或小程序进行交互的自定义调试工具。

这个包中包含有一个公共接口和 20 个公共类，公共接口和类一起可使我们实现一个程序调试器。调试接口采用客户/服务器体系结构模式。JVM 处在主持目标应用程序的服务器上，程序调试器是一个客户，作为界面控制目标应用程序，为了理解这种模型，我们按照以下的五个分类来讨论 JDB API 中的每个类：

- 客户/服务程序调试器管理
- 堆栈管理
- 特殊类型
- 线程管理
- 本地类型

程序调试器与正在运行的应用程序之间的交互方式是通过作为应用程序代理的一系列远程类型的类来实现的。一个代理的作用可看作是程序调试器与主机 JVM 之间的媒介人。我们也可以将代理想象为一个名人的公共交际代理人。公众从来不能直接与那个名人打交道，而只是与它的代理人联系。代理人这一角色的责任是发送信息给名人，并且作为中转站将名人的回应反馈给有关的公众。这正是 sun.tools.debug 包中所实现的类所做的事情。代理模型使类保持在较小的规模，并为与主机 JVM 交互提供一个清晰的接口。

> **注意**：JDB API 中的许多方法都抛出一般的 Exception 异常。不应把这种异常与我们想要或不想要捕获的异常相混淆。该 API 所抛出的异常一般情况下表示当 JVM 为一个调试器客户请求服务时发生在其内部的硬件错误情况。

在我们开始详细介绍 sun.tools.debug 包中各个类之前，首先了解一下这些类之间的层次结构可能会很有帮助。图 51.1 给出了 JDB API 类的层次关系图(松散 OMT 格式)。

51.1.1 客户/服务器程序调试器管理

建立在 JVM 上的调试工具最有趣的一个方面是它的客户/服务器本质。通过将 RemoteDebugger 类与我们的 DebuggerCallback 接口的实现结合在一起使用，可以在 JVM 为我们运行一个应用程序或小程序时，精确地控制它的各个方面。调试器客户与 JVM 通过一个私有

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

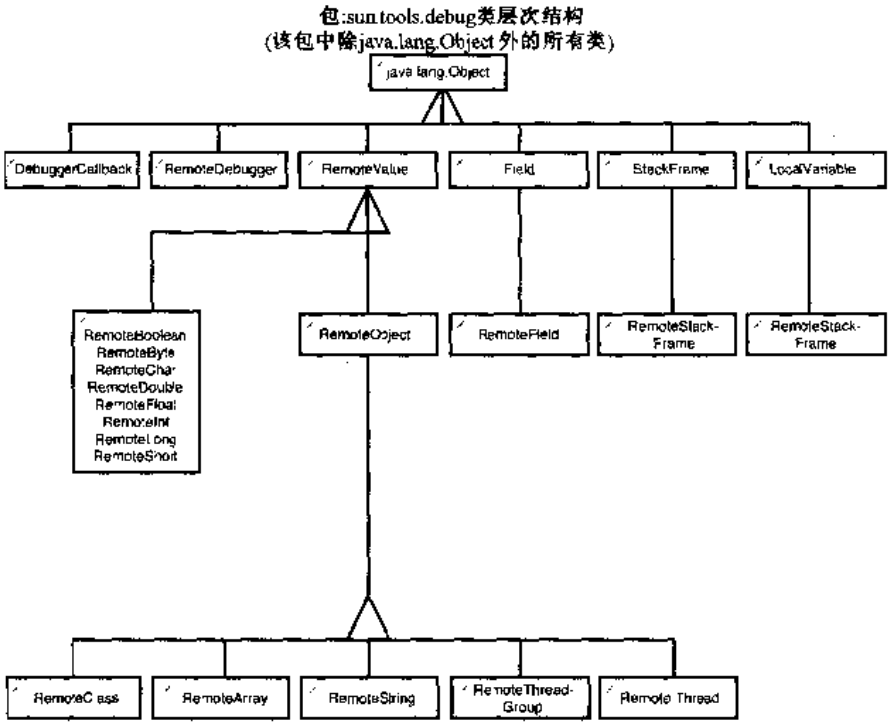


图 51.1　JDB API 的类层次图

TCP/IP 连接并使用一个独有的非文档协议(出于安全性的原因)进行通信。这一私有连接就是为什么 Debugger API 的源代码在 JDK 中找不到的原因。

> **注意**:还有两个类可以归入这一分类中:RemoteClass 和 RemoteThread 。RemoteClass 支持断点和异常管理,并提供有关类结构的描述信息。RemoteThread 通过对执行状态的操作支持对运行线程的控制。由于它们都是 RemoteObject 的子类,所以我们将在本章稍后的"特殊类型"一节中对其进行详细介绍。

DebuggerCallback 接口　DebuggerCallback 接口用于为 JVM 的调试器代理提供一种机制,用于当目标应用程序发生了应注意的事件时通知调试器客户。这些"事件"是使用 callback 方法处理的,该方法支持断点、异常、结束、线程死亡及控制台输出等。DebuggerCallback 接口的公共 API 在程序清单 51.1 中给出。

```
public interface DebuggerCallback {
    public abstract void printToConsole( String text ) throws Exception;
    public abstract void breakpointEvent( RemoteThread t ) throws Exception;
    public abstract void exceptionEvent( RemoteThread t,
  String errorText) throws Exception;
    public abstract void threadDeathEvent( RemoteThread t ) throws Exception;
    public abstract void quitEvent() throws Exception;
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 51.1 中列出了每个公共成员方法及其作用。

表 51.1 DebuggerCallback 接口的公共成员方法

名 称	描 述
printToConsole	当目标小程序向 System.out 或 System.err 发送输出时，或当主机 VM 中的调试器代理有消息时（特别是我们在 verbose 标志设置为 true(真)的情况下创建了自己的 RemoteDebugger 时）调用该方法
breakpointEvent	当到达目标应用程序断点时调用，t 是当到达断点时正在运行的线程
exceptionEvent	当目标应用程序抛出一个异常时发生，t 为异常发生时正在运行的线程，errorText 包含与异常一起发出的消息
threadDeathEvent	目标应用程序中线程 t 停止的信号
quitEvent	通知目标应用程序已经结束。这可能是调用 System.exit( )或从应用程序主线程返回的结果

RemoteDebugger 如果 RemoteCallback 接口是调试器的眼睛和耳朵，那么 RemoteDebugger 类就是调试器的嘴和手。RemoteDebugger 类是我们的控制“代理”，用来控制正在作为主机和主持正在被调试的应用程序或小程序的 JVM 实例。

要使用 RemoteDebugger 类，我们首先创建一个类来实现 RemoteCallback 接口，该类将成为一个 RemoteDebugger 实例的构造器的参数。（典型的情况下，调试器的主类将满足这种需求。）创建一个 RemoteDebugger 实例有两种方法：

- 连接到 JVM 的一个远程实例
- 有一个为我们所启动的 Java 命令实例

两种方式都以独立的方式运行，并使用 TCP/IP 与调试器进行内部“对话”。当创立完 RemoteDebugger 的一个实例后，我们将直接控制待调试的目标应用程序。然后就可以开始对自己的 RemoteDebugger 实例进行调用，以操作调试对话。

图 51.2 给出了执行时 JVM、RemoteDebugger 及 DebuggerCallback 之间的关系。

RemoteDebugger 类的公共 API 在程序清单 51.2 中给出。

```
public class RemoteDebugger {
    public RemoteDebugger( String host,
        String password,
        DebuggerCallback client,
        boolean verbose ) throws Exception;

    public RemoteDebugger( String javaArgs, DebuggerCallback client, boolean verbose ) throws Exception;

    public void addSystemThread()
    public void addSystemThread(thread t);
    public void close();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

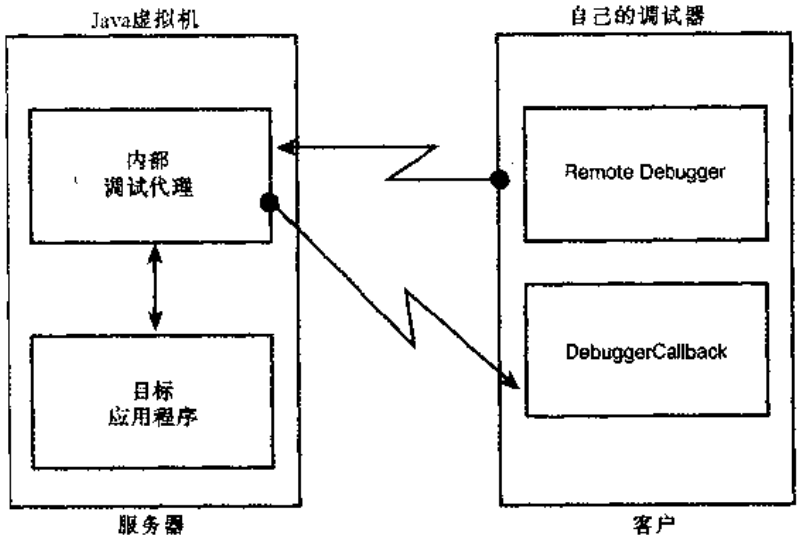


图 51.2　JVM、RemoteDebugger 及 DebuggerCallback 之间的关系

```
public RemoteObject get( Integer id );
public RemoteClass[] listClasses() throws Exception;
public RemoteClass findClass( String name ) throws Exception;
public RemoteThreadGroup[] listThreadGroups( RemoteThreadGroup tg )throws Exception;
public void gc( RemoteObject save_list[] ) throws Exception;
public void trace( boolean traceOn ) throws Exception;
public void itrace( boolean traceOn ) throws Exception;
public int totalMemory() throws Exception;
public int freeMemory() throws Exception;
public RemoteThreadGroup run( int argc, String argv[]) throws Exception;

public String[] listBreakpoints() throws Exception;
public String[] getExceptionCatchList() throws Exception;
public String getSourcePath() throws Exception;
public void setSourcePath( String pathList ) throws Exception;
```

表 51.2 中列出了每一个公共成员方法及其作用。

注意：当我们以 -debug 标志启动 java 或 appletviewer 时，会显示出一个特殊的口令，这个值必须用作 password 参数。

注意：切记，方法 trace 和指令 trace 的输出显示于主机 VM 的控制台上。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 51.2 RemoteDebugger 类公共成员方法

名 称	描 述
RemoteDebugger (String host, String password, DuggerCallback client, boolean verbose)	第一个构造器用于与一个现有的远程 JVM 连接。host 参数为运行 JVM 的目标计算机的 DNS 名字。password 是安全机制的一部分,用于安全地调试远程应用程序。client 是我们用来实现前面介绍的 RemoteDebugger 接口的对象。当 verbose 被设置为 true 时,将引起从主机的 JVM 向 client.printToConsole( ) 发送消息
RemoteDebugger (String javaArgs, DebuggerCallback client, boolean verbose)	第二个构造器与第一个很相似,只是它用于本地调试 Java 应用程序。对于大多数 GUI 应用程序来说,这可能是很好的。但是对于控制台应用程序来说使用这种方式调试是很困难的,因为调试目标的输出将与调试器本身的调试信息输出混杂在一起。javaArgs 参数应包含 java 命令的有效可选参数(不包括目标类),client 和 verbose 参数的作用与前面相同
addSystemThread( )	将正在调用的线程添加到未被调试器挂起的线程清单中。这些线程通常是调试器自己使用的线程
addSystemThread(Thread t)	将指定的线程 t 添加到调试器将不挂起的线程列表中
close( )	关闭远程目标应用程序或小程序以及主机 JVM
get(Integer id)	为 id 标识的对象返回一个代理。所返回的 RemoteObject 实例可能被强制转换为适当的类型。使用 ClassName 或 instanceof 测试其类型
ListClasses( )	返回驻留在主机 JVM 中的 RemoteClass 实例数组
FindClasses(String name)	搜索主机 JVM 中名为 name 的类。如果该类不在 VM 的类缓冲区中,则在目标计算机上继续搜索。如果仍未找到,则返回 null。可以传送名字的一部分,但这可能引起用户类和系统类之间使用相同名称的混乱
ListThreadsGroups	为包含在 tg 中的线程组返回一个(RemoteThreadGroup tg) RemoteThreadGroup 实例数组。如果 tg 为 null,将返回所有线程组
gc (RemoteObject save _ List[ ])	导致垃圾收集器在主机 JVM 上被运行,以释放该调试器实例所要求的所有对象。发送到 RemoteDebugger 的任何对象在该调试器完成之前或该调试器退出之前是不能作为垃圾被收集的。save _ list 用来保证任何仍在被本调试器实例检查的指定对象不作为垃圾被收集
trace(boolean traceOn)	在远程 JVM 中切换方法 trace 的标志状态。只有当使用以 javaArgs 作为第一个参数的构造器创建自己的 RemoteDebugger 实例时,或当远程调试主机是一个_ g 的变异时,这一命令才有效
itrace(boolean traceOn)	在远程 JVM 中切换指令 trace 的标志状态。只有当使用以 javaArgs 作为第一个参数的构造器创建自己的 RemoteDebugger 实例时,或当远程调试主机是一个_ g 的变异时,这一命令才有效
TotalMemory( )	返回主机 JVM 可用的内存总数
FreeMemory( )	返回主机 JVM 当前可用的内存总数
run (int argc, String argv [ ])	引起主机 JVM 装入并运行一个 Java 类。argv 是一个 String 数组,表示将要执行的各命令行。类名必须是数组的第一个元素。argc 是有效的元素个数。正在运行的类所在的 RemoteThreadGroup 实例在成功时被返回,否则返回 null
ListBreakpoints( )	返回主机 JVM 中当前活动的断点列表。断点列表的格式为 class.method _ name 或 class:line _ number
getExceptionCatchList( )	返回一个由异常名字所组成的数组,主机 JVM 将把这些异常名字像断点一样发送给调试器
GetSourcePath( )	返回在搜索与给定的类相关联的源文件时,主机 JVM 所使用的路径字符串。其格式与系统有关
setSourcePath (String pathList)	设置搜索与给定的类相关联的源文件时主机 JVM 所使用的路径字符串

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

51.1.2　特殊类型

在这一分类中类表示的是代理，这些代理使得我们能够在包含我们的目标的主机 JVM 中访问运行时实例数据。这些类之所以被认为是特殊的，是因为它们用于表示数据及控制元素的信息，而不是本地类型的信息。这些代理允许我们对加载到主机 JVM 中的对象和类进行检查并与之交互。例如，RemoteObject 及其子类表示的是目标小程序已经实例化的对象。

RemoteValue　RemoteValue 是一个抽象类，位于类树结构的根部，其作用是作为远程 JVM 的代理。该类实际上包含了一个由类树中接口以下的类实现的一个接口。由于这个类中包含抽象方法，所以不能对它进行显式实例化；相反，我们可以假设任何一个子类都一致并且安全地实现了 RemoteValue 方法。

RemoteValue 类的公共 API 在程序清单 51.3 中给出。

```
public class RemoteValue
        implements sun.tools.java.AgentConstants { // An undocumented
                                                   // interface containing static
                                                   // constants used internally
                                                   // by RemoteValue
        public String description();
        public static int fromHex( String hexStr );
        public final int getType();
        public final boolean isObject();
        public final boolean isString();
        public static String toHex( int n );
        public abstract String typeName() throws Exception;
}
```

表 51.3 中列出了每个公共成员方法及其作用。

表 51.3　RemoteValue 类的公共成员方法

名　称	描　述
description( )	返回本 RemoteValue 实例的一个常量描述
fromHex(String hexStr)	将 hexStr 中十六进制表示的一个数转换为一个整型值
getType( )	返回本地 RemoteValue 的类型的内部数字标识符。这个值主要由代理内部使用
isString( )	如果 RemoteValue 是一个字符串则返回 true（真）
isObject( )	如果 RemoteValue 是一个对象类型而不是一个本地语言类型（如 boolean），返回 true（真）
toHex(int n)	将一个整型值转换为 String 形式的十六进制表示
typeName( )	返回与本 RemoteValue 实例相关联的常量类型名

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

> **注意**:查看 CD-ROM 上的 ShowTypeCodes.java,这是一个显示这些值的简单工具。

RemoteField RemoteField 代理类与 RemoteValue 类相似,只是它是属于一个 RemoteClass 或 RemoteObject 实例的字段。该类提供了关于一个对象实例或类定义中一个字段的详细描述信息。一个字段可能是以下几种情况之一:一个实例变量、一个静态(类)变量、一个实例方法或一个静态(类)方法。RemoteField 类的公共 API 在程序清单 51.4 中给出。

程序清单 51.4 RemoteField 的部分公共 API

```
public class RemoteField
        extends sun.tools.java.Field // An undocumented class containing the
                                     // instance variables that hold the
                                     // representative values of the
                                     // RemoteField
implements sun.tools.java.AgentConstants {  // An undocumented interface
                                            // containing static
                                            // constants used internally by
                                            // RemoteField
        public String getModifiers();
        public String getName();
        public String getType();
        public boolean isStatic();
        public String toString();
        public String getTypedName()
}
```

表 51.4 中列出了每个公共成员方法及其作用。

表 51.4 RemoteField 类的公共成员方法

名 称	描 述
getModifiers( )	以常量形式(例如,public 或 private)返回关于本 RemoteField 的访问修饰符
getName( )	返回常量字段名
getType( )	以 String(如 int、boolean 或 java.lang.String)返回本字段的类型
isStatic( )	如果字段被指定为静态的(static)则返回 true(真)(见前面讨论的 getModifiers( ))
toString( )	以 String 形式返回本字段的值,与其本地类型相反
getTypeName( )	返回描述本字段及其类型名的字符串

RemoteObject RemoteObject 是允许我们与主机 JVM 中一个类的实例进行接口的一个代理。它用于访问关于所请求的对象实例的详细信息,包括它的类、字段信息及字段值。我们使用 RemoteObject 可以枚举一个对象实例的数据。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

警告:我们应当注意到,当从主要 JVM 请求 RemoteObject (或其任何子类)实例时,主机 JVM 将把这些实例保存在内存的一个非垃圾收集区域。对于任何不再需要的 RemoteObject 实例,调试器应调用(周期性地或使用命令) RemoteDebugger.gc( )方法进行释放。

RemoteObject 类的公共 API 在程序清单 51.5 中给出。

程序清单 51.5　RemoteObject 类的公共 API

```
public class RemoteObject
    extends RemoteValue {

    public String description();
    public final RemoteClass getClazz();
    protected void finalize() throws Exception;
    public RemoteField getField( int slotNum ) throws Exception;
    public RemoteField getField( String name ) throws Exception;
    public RemoteValue getFieldValue( int slotNum ) throws Exception;
    public RemoteValue getFieldValue( String name ) throws Exception;
    public RemoteField[] getFields() throws Exception;
    public final int getId();
    public String toString();
    public String typeName() throws Exception;
}
```

表 51.5 中列出每一个公共成员方法及其作用。

表 51.5　RemoteObject 类的公共成员方法

名　称	描　述
description( )	覆盖 RemoteValue.description( )方法
getClazz( )	返回对应于本对象实例的一个 RemoteClass 实例
getField( )	这是一个重载的方法,返回一个 RemoteField 实例。该实例基于表示对象中本字段物理位置的插槽编号,或者基于字段的常量名。如果字段不存在(slotNum 或 name 无效),抛出 Exception 异常。如果 name 没有找到,则返回 null
getFieldValue( )	这是一个重载方法,以 RemoteValue 实例返回本字段的值。搜索或者基于对象中表示本字段物理位置的插槽编号,或者基于字段的常量名。如果字段不存在(slotNum 或 name 无效),抛出一个 Exception 异常。如果 name 没有找到,则返回 null
getFields( )	返回 RemoteField 实例的一个列表表示所有在本对象的类中定义的非静态实例变量
getId( )	返回在主机 JVM 中用来唯一标识本对象的实例 ID
toString( )	以 String 形式返回本对象的一个表示。这完全取决于正在使用的对象实例的类型
typeName( )	覆盖 RemoteValue.typeName( )
finalize( )	包含本对象被垃圾收集器收集时需要执行的代码

RemoteClass　RemoteClass 代表了 sun.tools.debug 包中较大的 API 之一,提供了关于类

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

定义各个方面的详细信息，包括其超类、字段(静态的和实例的)、所实现的接口以及方法。由于它是 RemoteObject 的后代，切记当调试器结束时要在某一点上对实体调用 gc( )。我们可以从 RemoteDebugger、RemoteObject 及 RemoteStackFrame 实例中提取 RemoteClass 的实例。

RemoteClass 类的公共 API 在程序清单 51.6 中给出。

```
public class RemoteClass extends RemoteObject {

    // Descriptive methods:
    public String description();
    public RemoteObject getClassLoader( ) throws Exception;
    public RemoteField getField( int slotNum ) throws Exception;
    public RemoteField getField( String name ) throws Exception;
    public RemoteValue getFieldValue( int slotNum ) throws Exception;
    public RemoteValue getFieldValue( String name ) throws Exception;
    public RemoteField[] getFields( ) throws Exception;
    public RemoteField getInstanceField( int slotNum ) throws Exception;
    public RemoteField[] getInstanceFields( ) throws Exception;
    public RemoteClass[] getInterfaces( ) throws Exception;
    public RemoteField getMethod( String name ) throws Exception;
    public String[] getMethodNames( ) throws Exception;
    public RemoteField[] getMethods( ) throws Exception;
    public String getName( ) throws Exception;
    public RemoteField[] getStaticFields( ) throws Exception;
    public RemoteClass getSuperclass( ) throws Exception;
    public String getSourceFileName( );
    public boolean isInterface( ) throws Exception;
    public String toString( );
    public String typeName( ) throws Exception;

    // Debugging management methods:
    public InputStream getSourceFile( ) throws Exception;
    public void catchExceptions( ) throws Exception;
    public void ignoreExceptions( ) throws Exception;
    public String setBreakpointLine( int lineNo ) throws Exception;
    public String setBreakpointMethod( RemoteField method ) throws Exception;
    public String clearBreakpoint( int pcLocation ) throws Exception;
    public String clearBreakpointLine( int lineNo ) throws Exception;
    public String clearBreakpointMethod(RemoteField method) throws Exception;
    public int[] getLineNumbers( ) throws Exception
    public int getMethodLineNumber(int index)throws IIndexOutOfBoundsException,
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
NoSuchLineNumberException, Exception
  public int getMethodLineNumber(String name) throws NoSuchMethodException,
NoSuchLineNumberException, Exception
}
```

表 51.6 和表 51.7 中列出了每个公共成员方法及其作用。

表 51.6　RemoteClass 类的描述性公共成员方法

名　称	描　述
description	覆盖 RemoteObject.description( )
getClassLoader( )	返回表示本类的类加载器的一个 RemoteObject 实例
getField( )	这是一个重载方法，它基于本对象中表示本字段物理位置的一个插槽编号或基于该字段的常量名为一个静态成员返回一个 RemoteField 实例。如果该字段不存在(slotNum 或 name 无效)，抛出一个 Exception 异常。如果没有找到 name，则返回 null
getFieldValue( )	这是一个重载方法，以 RemoteValue 实例返回一个静态字段的值。搜索基于对象中表示本字段物理位置的插槽编号，或基于字段的常量名。如果字段不存在(slotNum 或 name 无效)，抛出一个 Exception 异常。如果没有找到 name，则返回 null
getFields( )	覆盖 RemoteObject.getFields( )方法，但返回一个表示本类中所有可用的静态字段的 RemoteField 实例数组
getInstanceField( )	返回一个 instance 字段描述，作为 RemoteField 的一个实例。搜索是基于对象中表示本字段物理插槽编号。如果字段不存在(slotNum 无效)，抛了一个 Exception 异常。注意当在此上下文中被调用时，字段中不存在实例数据
getInstanceFields( )	返回一个 RemoteField 实例数组，表示本类中可以使用的所有实例字段。注意，在此上下文中被调用时，字段中不存在实例数据
getInterfaces( )	返回一个 RemoteClass 实例数组，表示本类中实现的每一个接口
getMethod( )	使用 name 查找并返回一个描述指定方法签名的 RemoteField 实例
getMethodNames( )	返回一个 String 数组，其中包含本类中所实现的所有方法的名字
getMethods( )	返回一个 RemoteField 实例数组，表示本类中实现的所有方法
getName( )	返回一个包含本类名字的字符串
getSourceFileName( )	返回包含用于编译本类的 Java 源语句的文件名。这里只包括文件的基本名和扩展名(格式与操作系统有关)，不包括任何路径信息(例如，MyClass.java)
getStaticFields( )	返回一个 RemoteField 实例数组，表示本类中可以使用的所有静态字段
getSuperclass( )	为本类的超类返回一个 RemoteClass 实例。如果没有指定超类(类定义中没有使用 extend 子句)，返回 java.lang.Object 的一个实例
isInterface( )	如果该 RemoteClass 实例表示一个接口而不是一个类定义，则返回 true(真)值
toString( )	覆盖 RemoteObject.to.String( )方法
typeName( )	覆盖 RemoteObject.typeName( )方法
getLineNumbers	以数组形式返回具有与之相关联代码的类中的源文件行号
getMethodLineNumber	该方法返回由一个字符串或索引号指定的方法的第一行

RemoteArray　Java 中的数组都是对象，这适用于调试器观察数组的方式。所以，我们有

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

一个称为 RemoteArray 的特殊类型。这种类型允许我们询问主机 JVM 中一个数组运行时的实例。在 RemoteArray 类中存在的区别之一是不存在一种直接获得其 RemoteField 信息的手段。所以在调试器中，我们将使用 RemoteObject.getField( )方法来获取这一信息。然后使用 RemoteObject.getFieldValue( )方法来访问数组的实际元素，并将其返回类型强制为 RemoteArray 类型。

表 51.7 RemoteClass 调试管理公共成员方法

名 字	描 述
getSourceFile( )	返回一个 InputStream 实例，我们可以用这个实例来显示源文件中用来生成本类的代码行(如果可以获得，返回一个非零值)。该方法的目的是在我们的调试器实现中提供一个 list 类型的命令，或者是提供交互式源程序级调试。所返回的 InputStream 在使用之前一般都强制转换为 DataInputStream
catchExceptions( )	当本类的一个实例被抛出时通知主机 JVM 将控制返回给调试器。如果本类不是 Exception 的一个子类，则抛出一个 ClassCastException 异常。这将使异常对调试器来说，看上去像是一个断点，并且导致调用 DebuggerCallback.exceptionEvent( )
ignoreExceptions( )	通知主机 JVM 在本异常被抛出的情况下不要向调试器发出信号。主机 JVM 仍要抛出该异常，但不是向我们抛出。如果本类不是 Exception 的一个子类，则抛出一个 ClassCastException 异常。在实际使用时，假设 castException( )已经为本类调用过了
setBreakpointLine( )	允许调试器基于 lineNo 中指定的源文件行号设置断点。如果 lineNo 不在范围之内，或发生其它一些错误，返回一条消息。否则，成功地返回一个空字符串("")。如果设置成功，当遇到断点时调用 DebuggerCallback.breakpointEvent( )
setBreakpointMethod( )	允许调试器基于 RemoteField 的一个实例设置一个断点，该实例包含对本类中一个方法的引用。断点设置在该方法的第一行上。如果由于某些原因所指定的 method 为非法的，成功地返回一个空字符串("")。如果设置成功，当遇到断点时，调用 DebuggerCallback.breakpointEvent( )方法
clearBreakpoint( )	使用一个合法的程序计数器(Program Counter，PC)值作为在 pcLocation 中指定的值，调试器可以删除断点。如果由于某种原因，pcLocation 无效，返回一个错误信息。否则，返回一个空字符串("")。该方法只有有限的值，因为文档中没有以这种方式指定断点的方法
clearBreakpointLine( )	删除以前为 lineNo 指定的断点。如果由于某种原因 lineNo 非法，返回一个错误信息。否则，返回一个空字符串("")
clearBreakpointMethod( )	删除以前为 method 指定的断点。如果由于某种原因 method 非法，返回一个错误信息。否则，返回一个空字符串("")

RemoteArray 类的公共 API 在程序清单 51.7 中给出。

```
public class RemoteArray extends RemoteObject {
    public String arrayTypeName( int type );
    public String description();
    public final RemoteValue getElement( int index ) throws Exception;
    public final int getElementType() throws Exception;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public final RemoteValue[] getElements() throws Exception;
    public final RemoteValue[] getElements( int beginIndex, int endIndex)throws Exception;
    public final int getSize();
    public String toString();
    public String typeName();
}
```

表 51.8 中列出了每一个公共成员方法及其作用。

表 51.8　RemoteArray 类中的公共成员方法

名　　称	描　　述
arrayTypeName( )	返回一个字符串，其中包含了数组元素的类型。type 参数通过调用随后所定义的 getElementType( ) 来提供。对于任何一个 java.lang.Object 的子类的类型，返回字符串 Object。我们需要对由 getElement( ) 或 getElements( ) 所返回的 RemoteValue 实例使用 RemoteValue.typeName( ) 方法，以获得实际对象类的名字
description	覆盖 RemoteObject.description( ) 方法
getElement( )	返回一个 RemoteValue 实例，其中包含索引值为 index 的数组元素的值。如果 index 下标超出数组的界，getElement 抛出一个 ArrayIndexOutOfBoundsException 异常
getElementType( )	返回一个数字标识符(内部定义的)，该标识符表示数组中所包含的元素的数据类型。使用前面定义的 arrayTypeName( ) 方法，可获得与这种类型相关的常量值
getElements( )	这是一个重载方法，用于返回 RemoteValue 实例的全部或一个子集所组成的数组。如果不指定参数，则返回所有值；如果指定了 beginIndex(从 0 开始)和 endIndex(最大为 getSize( ) - 1)，则返回指定范围的 RemoteValue 实例的子集。如果索引值非法，则抛出一个 ArrayIndexOutOfBoundsException 异常
getSize( )	返回包含在这一数组实例中元素的实际个数
toString( )	覆盖 RemoteObject.toString( ) 方法
typeName( )	覆盖 RemoteObject.typeName( ) 方法

RemoteString　RemoteString 类是“特殊类型”中最后一个类。之所以说它是特殊的是因为它是 RemoteObject 的一个子类，但是由于编译器处理它的方式的原因，它更接近于一个本地类型。这个类非常简单——对一个字符串可做的操作就是显示它的内容。

RemoteString 类的公共 API 在程序清单 51.8 中给出。

程序清单 51.8　RemoteString 类的公共 API

```
public class RemoteString extends RemoteObject {
    public String description();
    public String toString();
    public String typeName();
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 51.9 中列出了每一个公共成员方法及其作用。

表 51.9 RemoteString 类的公共成员方法

名 称	描 述
description	覆盖 RemoteObject.description( )方法,并返回字符串对象值或 null 常量
toString	覆盖 RemoteObject.toString( )方法,并返回字符串对象值或 null 常量
typeName	覆盖 RemoteObject.typeName( )方法

51.1.3 本地类型

本地类型的类都是基于 RemoteValue 的代理,用于检查基于非对象的本地类型的字段类型和堆栈变量。所有的本地类型共享同一个接口,允许通过 RemoteValue 抽象类型,来完成多态性使用。

表 51.10 中描述了该 API 所支持的本地类型。

表 51.10 RemoteXXX 类所支持的本地类型

本地类型	RemoteXXX 类
boolean	RemoteBoolean
byte	RemoteByte
char	RemoteChar
double	RemoteDouble
float	RemoteFloat
int	RemoteInt
long	RemoteLong
short	RemoteShort

本地类型类所共享的公共 API 在程序清单 51.9 中给出。< RemoteXXX > 标记中的 XXX 部分可以用任何以大写字母开头的本地类型所替代(例如,对 boolean 为 RemoteBoolean),如表 51.10 中的描述所示。

```
public class < RemoteXXX > extends RemoteValue {
    public < native type > get();
    public String toString();
    public String typeName();
}
```

表 51.11 中列出了每个公共成员方法及其描述。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 51.11　RemoteXXX 类的公共成员方法

名　称	描　述
get	以一个本地值的形式返回 RemoteXXX 类中所包含的值。程序清单 51.9 中所给出的 < native type > 指示符，可根据当前实例所代表的实际 RemoteXXX 类，使用表 51.10 中“本地类型”一列中的任何类型来替代。例如，对 RemoteBoolean 类，从其 get( ) 方法返回一个实际的布尔类型值
toString	覆盖 RemoteObject.toString( )方法
typeName	覆盖 RemoteObject.typeName( )方法

51.1.4　堆栈管理

当我们在调试器中获得一个点，以便在这一点可以停止程序的执行并检查各种情况时，堆栈就变成了一个很重要的问题。在 JVM 中，描述正在执行的方法当前状态的所有信息都存放在一个堆栈帧中。堆栈帧包括方法参数、局部变量、程序计数器(PC)、方法名等各项。RemoteStackFrame 表示了一个正在运行的 Java 方法所有执行时的特性，它是从调试器控制本单元的代理。反过来，RemoteStackFrame 通过 RemoteStackVariable 实例，向我们提供物理地驻留在这一堆栈帧中的数据信息。这些 RemoteStackVariable 实例是当前上下文的方法可以使用的实际变量的代理。当前上下文可定义为活动的堆栈帧方法。

StackFrame 类的公共 API 在程序清单 51.10 中给出。

```
public class StackFrame {
    public StackFrame();
    public String toString();
}
```

RemoteStackFrame　RemoteStackFrame 类是一个代理，它允许我们与主机 JVM 中挂起线程的堆栈帧进行交互。RemoteStackFrame 实例基本上描述了自己的执行状态并枚举其变量。调试器使用这个类与其它的调试器类一起，显示上下文中方法的状态。

RemoteStackFrame 类的公共 API 在程序清单 51.11 中给出。

```
public class RemoteStackFrame extends StackFrame {
    public int getLineNumber();
    public RemoteStackVariable getLocalVariable( String name )
            throws Exception;
    public RemoteStackVariable[] getLocalVariables() throws Exception;
    public String getMethodName();
    public int getPC();
    public RemoteClass getRemoteClass();
}
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 51.12 中给出了每个公共成员方法及其作用。

表 51.12 RemoteStackFrame 类的公共成员方法

名 称	描 述
getLineNumber( )	返回相对于源文件开始处的行号，该行号与挂起线程的本堆栈帧的当前位置相关联
getLocalVariable( )	返回与本 RemoteStackFrame 实例中的 name 相关联的 RemoteStackVariable 实例
getLocalVariables( )	返回一个在 RemoteStackFrame 实例中可以使用的 RemoteStackVariable 实例数组
getMethodName( )	返回一个字符串，其中包含本 RemoteStackFrame 实例所表示的方法的名字
getPC( )	返回本 RemoteStackFrame 实例表示的方法中第一个字节码相对应的 JVM 的 PC。PC 像是 Java 字节码流的一个指针，当 VM 对每一个字节码进行解释时 PC 随之移动
getRemoteClass( )	返回一个 RemoteClass 实例，该实例定义本 RemoteStackFrame 实例所表示的方法

RemoteStackVariable RemoteStackVariable 类是一个代理，它使得我们可以访问包含在一个 RemoteStackFrame 实例中的值。方法的参数及局部变量全都以存放状态、变量标识及其当前值的 RemoteStackVariable 实例返回。

RemoteStackVariable 类的公共 API 在程序清单 51.12 中给出。

```
public class RemoteStackVariable extends LocalVariable {
    // This is a private class that contains the data
    // items that are exposed via Remote -
    // StackVariable methods
    public String getName();
    public RemoteValue getValue();
    public boolean inScope();
}
```

表 51.13 中列出了每个公共成员方法及其作用。

表 51.13 RemoteStackVariable 类的公共成员方法

名 称	描 述
getName	返回包含本 RemoteStackVariable 实例的常量名字的一个字符串
getValue	返回包含此时本变量数据值的一个 RemoteValue 实例。该对象可以强制转换成一个适当的 RemoteValue 的子类
isScope	如果 RemoteStackVariable 实例当前处于其作用范围之内，则返回 true(真)。如果定义 RemoteStackVariable 的块不在上下文中，则称该 RemoteStackVariable 不在其作用范围内——例如，for 循环结构定义的一个计数器或 catch 语句中定义的一个异常

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

51.1.5　线程管理

在编写 Java 应用程序时更具有挑战性的任务之一是如何调试线程。对 Java 开发人员来说，幸运的是 Sun 公司提供了帮助调试多线程应用程序的特殊类。Java 的线程管理器基于两种结构：线程组和线程本身。Java 使用线程组来帮助分类和隔离相关的独立执行路径或线程集合。

多线程应用程序的调试工具允许我们检查和控制线程组和独立线程的执行。这种控制和执行是由 RemoteThreadGroup 和 RemoteThread 类完成的。

调试多线程应用程序的一个复杂问题是如何操纵处于活动状态的独立线程。当我们在一个给定类的方法中设置一个断点后，经过这一执行路径的所有线程都将中断。实际上所发生的事情是当前线程中断，而其它沿着同一路径的线程则挂起。在这种情况下，我们可以使用 RemoteThreadGroup 和 RemoteThread 类来恢复其它的线程，与此同时我们可以继续调试（step、examine 等）所关心的单独线程。记住，调试一个多线程应用程序的技术与调试一个单线程应用程序是一样的。

调试多线程应用程序也可能是很困难的，这取决于这些线程如何共享数据的逻辑关系。Java 语言提供了许多内置的工具，控制线程对共享数据的并发访问。synchronized 关键字、Objcet 类的 wait( ) 和 notify( ) 方法、以及 Thread 类的 sleep( ) 和 yield( ) 方法等，都提供了可以帮助我们进行逻辑设计的特性，以便使含有线程的应用程序中数据共享可以安全地实现。调试工具可以帮助我们识别出缺少这些并发机制的逻辑区域。

以下的两个主题——RemoteThreadGroup 和 RemoteThread ——讲述了允许我们进行这些操作的代理类。

RemoteThreadGroup　RemoteThreadGroup 类是运行于主机 JVM 的一个实际线程组实例的代理。所以，它代表了一个容器，其中含有 RemoteThread 实例以及嵌入的 RemoteThreadGroup 实例。对 RemoteThreadGroup 的接口则相当简单，它提供了检索远程线程列表及停止包含在当前线程组中所有线程和线程组执行的功能。

RemoteThreadGroup 类的公共 API 在程序清单 51.13 中给出。

```
public class RemoteThreadGroupextends RemoteObject {
    public String getName( ) throws Exception;
    public RemoteThread[ ] listThreads( boolean recurse ) throws Exception;
    public void stop( ) throws Exception;
}
```

表 51.14 中列出了每个公共成员方法及其作用。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 51.14 RemoteThreadGroup 类的公共成员方法

名 称	描 述
getName	返回当前 RemoteThreadGroup 实例的名字
listThreads	返回当前 RemoteThreadGroup 实例中存在的 RemoteThread 实例的一个数组。如果 recurse 设置为 true,则遍历所有嵌入的 RemoteThreadGroup,并返回它们的成员 RemoteThread 实例
stop	停止属于本线程组的所有线程的执行。如果我们正在调试一个多线程应用程序,有多个线程实例运行于相同的执行路径,那么这个方法非常有用。如果不使用这个方法,可使用 ListThreads( ),手工选择想要停止的线程

RemoteThread RemoteThread 类是多线程调试 Java 应用程序的核心。它提供了一个线程停止执行后控制线程执行的接口。执行停止的原因可能是遇到了一个断点,也可能是对 suspend( )或 stop( )的一个显式调用。当线程的执行挂起之后(由于某种原因),我们可以检查线程的当前状态、单步执行线程、操作堆栈帧以及检查范围内的任何变量。在调试器中实现 RemoteThread 的使用意味着我们现在已经拥有了管理远程线程的执行所需的一切能力。

由于 RemoteThread 类的 API 非常庞大,我们将其分为三类:

- 基本线程控制。控制当前线程实例整体执行状态的方法。
- 执行路径控制。当一个线程挂起(不论是由于手工操作、由于一个异常、还是由于遇到了一个断点)之后控制程序流程的方法。
- 堆栈帧控制。操作堆栈帧、检查当前帧中局部变量和参数的方法。

RemoteThread 类的公共 API 在程序清单 51.14 给出。

```
public class RemoteThreadextends RemoteObject {

    // Basic Thread Control
    public String getName() throws Exception;
    public String getStatus() throws Exception;
    public boolean isSuspended();
    public void resume() throws Exception;
    public void stop() throws Exception;
    public void suspend() throws Exception;
    // Execution path control
    public void cont() throws Exception;
    public void next() throws Exception;
    public void step( boolean skipLine ) throws Exception;

    // Stack frame control
    public void down( int nFrames ) throws Exception;
    public RemoteStackFrame[] dumpStack() throws Exception;
    public RemoteStackFrame getCurrentFrame() throws Exception;
    public int getCurrentFrameIndex();
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public RemoteStackVariable getStackVariable( String name )
        throws Exception;
    public RemoteStackVariable[] getStackVariables() throws Exception;
    public void resetCurrentFrameIndex();
    public void setCurrentFrameIndex( int iFrame );
    public void up( int nFrames ) throws Exception;
}
```

表 51.15 中列出了每一个与基本线程控制有关的公共成员方法及其作用。

表 51.15　RemoteThread 类关于基于线程控制的公共成员方法

名　称	描　述
getName( )	返回包含本 RemoteThread 实例的名称的一个字符串
getStatus( )	返回包含本 RemoteThread 实例常量的一个字符串
isSuspended( )	如果本 RemoteThread 实例挂起则返回 true(真)
resume( )	从当前程序计数器恢复本 RemoteThread 实例的执行。它假设本线程当前是挂起的
stop( )	结束本 RemoteThread 实例的执行。当停止线程执行之后，不能再恢复线程的执行，但可以检查当前堆栈帧
suspend( )	在本 RemoteThread 实例的当前所在位置挂起它的执行。这与线程实例遇到断点时类似：当执行挂起时，我们可以使用执行路径控制方法单步执行线程。该线程可以使用 resume( )方法恢复其执行

表 51.16 中列出了每一个与执行路径控制有关的公共成员方法及其作用。

表 51.16　RemoteThread 类关于执行路径控制的公共成员方法

名　称	描　述
cont( )	从断点处恢复当前 RemoteThread 实例。如果线程是挂起的，则使用 resume( )而不是 cont( )
next( )	执行当前 RemoteThread 实例中源程序的下一行，并且不“步入”这一过程中所调用的任何方法。这也就是，它执行中间调用方法但不提供停止执行和检查其变量等的机会。如果线程未挂起或正在处理一个断点，则该方法抛出一个 IllegalAccessError 异常。另外如果不存在这个类的行号信息，next 操作与下面的 step( )方法所做的操作类似
step( )	执行下一指令或当 skipLine 为 true 时执行到下一行。在这一点上执行 step(false)将把 PC 置于 evaluateCounter( )的第一条指令，而调用 next( )执行对 evaluateCounter( ) 的调用并使 PC 指向第三行的第一条指令

注意：与 next( )不同，step( )方法则“步入”执行过程中遇到的任何中间调用方法。下面的源代码行是一个例子(此处 PC 是当前程序计数器)：

```
1: myCouter + = 1;
2: evaluateCounter(myCounter);
3: System.out.println("myCouter:" + myCounter);
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 51.17 中列出了每一个与堆栈帧控制有关的公共成员方法及其作用。

表 51.17 RemoteThread 类中关于堆栈帧控制的公共成员方法

名 称	描 述
down	向下移动当前上下文中关于本 RemoteThread 实例的堆栈帧 nFrames 层。一般用于执行 up( ) 方法之后,以便"走"回所调用堆栈帧。该命令与 up( )结合起来使用,可帮助实现一个交互式调用堆栈窗口。如果本 RemoteThread 实例未挂起或处于一个断点上,则抛出一个 IllegalAccessError 异常。另外,如果 nFrames 太大(例如,试图转移到执行堆栈的底层帧之外),则产生一个 ArrayOutOfBounds 异常
dumpStack( )	返回一个 RemoteStackFrame 实例数组,表示执行堆栈帧直到并包括当前帧。为了显示调用堆栈,可以通过 RemoteStackFrame 实例数组进行迭代并调用其 toString 方法
getCurrentFrame( )	返回当前帧的 RemoteStackFrame 实例
getCurrentFrameIndex( )	返回执行堆栈中当前 RemoteStackFrame 的索引
getStackVariable( )	返回当前堆栈帧中与 name 有关联的 RemoteStackFrame 实例
getStackVariables( )	返回包含在当前堆栈帧中的 RemoteStackFrame 实例数组。这些实例表示方法的参数及局部变量(不管这一点是否在其范围之内)
resetCurrentFrameIndex( )	恢复堆栈帧到其调用 up( )、down( ) 或 setCurrentFrameIndex( )之前的状态
SetCurrentFrameIndex( )	将位于 iFrame 层的堆栈帧作为主机 JVM 的当前栈帧建立
up( )	向上移动当前上下文中 RemoteThread 实例的栈帧 nFrames 层。一般情况下用于一个断点之后,以便"走"到上面的调用栈帧。当这个方法与 down( )方法结合使用时,可以帮助我们实现一个交互式调用堆栈窗口。如果本 RemoteThread 实例未挂起或正处于一个断点上,则抛出一个 IllegalAccessError 异常。另外,如果 nFrames 太大(例如,试图转到执行堆栈中的顶层帧之外去),则抛出一个 ArrayOutOfBounds 异常

51.1.6 总结

到现在,我们可能会问"我们如何利用所有这些先进的技术呢?"首先,我们可以只是简单地使用下一节中将介绍的 JDB。或者我们可能会考虑以 JDB 作为一个样本应用程序,并编写自己的调试器。对这种情况,下面的部分将向我们提供使用 sun.tools.debug 包中的类的一些基本指导。

在本书所附的光盘中有一个名为 DebuggerSkeleton.java 的文件。这是一个基于 sun.tools.debug 包的调试器外壳。DebuggerSkeleton.java 展示了如何通过实现 DebuggerCallback 接口和实例化 RemoteDebugger 类的一个实例,来开始编写调试器。

我们可以以下面的步骤作为指导,使用 JDB API 实现自定义的调试工具:

1.创建一个实现 DebuggerCallback 接口的基类。

2.在基类中创建一组针对状态的实例变量,用来存放诸如 RemoteDebugger 实例、当前线程组以及当前线程等项目(这也是 JDB 使用的相同模式)。

3.使用两个构造器中任何一个,创建一个 RemoteDebugger 实例。所选择的构造器依我们是要远程调试、本地调试或二者皆有的具体需求而定。

4.如果我们正在创建一个基于命令行的调试器,可启动一个命令循环以接受交互调试命令;如果我们正在开发一个基于 GUI 的调试器,那么命令逻辑一般通过按钮和菜单事件来执行。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

5.我们或许会按照如下的方式组织自己的命令处理,如表 51.18 所示。

表 51.18　命令处理组织

分　类	描　述
一般	这些命令处理调试器控制的一般流程,可使用 RemoteDebugger 类实例来处理这些命令。考虑选择诸如上下文命令(设置当前线程组和线程)、内存命令、跟踪等作为这一分类的潜在命令
信息	这些命令显示当前调试目标的信息。可使用 RemoteObject、RemoteClass 以及 RemoteStackFrame 的实例,显示对象、类方法、变量以及源程序行等
断点	这些命令用于设置或重新设置断点和表达式。可使用 RemoteClass 和 RemoteThread 类中的方法来实现这些功能
执行	这是一些可以在遇到断点、线程挂起或产生异常时使用的命令。可使用 RemoteClass 和 RemoteThread 来处理这些请求

51.2　深入探讨 JDB

现在我们已经对 JDK 中的底层调试工具有了一个很好的了解,可以对 JDB 进行检验。JDB 实际上提供了两种服务:

- 作为一个 Java 程序的调试工具
- 作为样本应用程序为使用 JDB API 中的类服务

我们此处关于 JDB 的讨论包括所有命令的详细介绍和使用 JDB API 时其主要部分的描述。

51.2.1　基本结构

作为一个应用程序,JDB 的模式采用了大多数 UNIX 系统下可找到的 DBX 调试器的模式。这是一个面向命令行的调试器,允许我们与正在运行的应用程序通过输入类似于英语的命令进行交互,以检查和控制程序的执行状态。这些命令允许我们检查变量、设置断点、控制线程以及查询主机 JVM 中加载的类信息。我们也可能要让主机 JVM 提前加载一些类,以便在执行它们之前在它们的方法中设置一些断点。

为了全面地理解 JDB 的结构,在阅读这一部分的内容时,如果能够打印出或访问到它们的源代码,将会很有帮助。在安装 JDK 时,安装 JDK 的根目录下(Windows 版本下一般为 \JAVA)有一个相对较小的 ZIP 文件名为 SRC.ZIP。该文件包含了所有公共文档中 Java 类的源程序文件,包括作为 JDB 基础的类。但是对 SRC.ZIP 文件必须使用 Use Directory Names(使用目录名)选项进行解压缩,以保留原来的源文件之间的树型结构。这些源文件的树结构遵循了 JDK 中的命名习惯。

例如,如果我们解压缩 SRC.ZIP 文件到 \JAVA 下的一个目录 SRC 下,则在 \JAVA\SRC 目录下应该可以找到两个称为 \JAVA\SRC\JAVA 和 \JAVA\SRC\SUN 的子目录。它们分别以 java.* 和 sun.* 集合表示 Java 源文件。

JDB 的源是以一个称为 TTY 或 Sun.tools.ttydebug.TTY 的类为基础的。其源文件(假设目录结构同前)将在目录 \JAVA\SRC\SUN\TOOLS\TTYDEBUG\TTY.JAVA 中。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

当查看 TTY.java 时，我们可能注意到的第一件事情就是 TTY 是一个简单的类，它是从 Object 中派生出来的，就像没有使用 extends 子句的类一样。但是，它确实像我们在前面“总结”一节中提到的那样，实现了 DebuggerCallback 接口。

应该注意到其中有一些实例变量可以用来帮助支持应用程序的一致性。特别地，为了维护当前执行的应用程序内的上下文关系需要一个对 RemoteDebugger 实例的引用(debugger)，一个 RemoteThreadGroup(currentThreadGroup)以及一个 RemoteThread(currentThread)。这将有助于从挂起的线程和方法中查询信息的大部分命令的实现。在几个私有方法之后，我们将看到定义在 DebuggerCallback 中的一些方法，这些方法允许 TTY 通过实现这一接口完成其约定。

现在可以跳到源文件的末尾部，来看看其真实的结构。它是以一个 main 方法开始，该方法在装入并运行 TTY.class 时第一个被调用。main 方法主要是分析、收集和检查所有命令行参数，如果一切正常则创建一个 TTY 类的实例。这一过程的其余部分则主要是针对 TTY 的自定义构造器的。

TTY 中的唯一构造器使用了七个参数，这些参数说明了以下这些内容：

- 远程 JVM 的定位和连接信息
- 需要加载的类文件(可选)
- 调试器和远程 JVM 的输出文件
- 一个 Boolean 标志，用来说明在调试对话处于活动状态时，是否需要远程 JVM 返回大量的信息型消息。

在构造器中，当创建远程调试器实例之后，如果指定的话，则加载初始类。创建一个 RemoteDebugger 实例，实际上导致一个 JVM 在远程系统启动(即使没有指定启动类)。在这之后，是对一个输入命令文件的检查。最后，命令处理循环开始。

命令处理循环的功能实际上由一个名为 executeCommand 方法来实现，该方法需要输入命令行有一个标志。executeCommand 方法只是由一系列平铺式的 if - else - if 语句组成，这些语句检查所输入参数的第一个标志，看它是否与预先定义的由 JDB 支持的命令集中的某个命令相匹配。如果匹配，与该命令相关联的辅助方法开始执行。否则，将显示一个适当的错误信息(“Huh? Try help...”)。

现在我们对 JDB 的一般结构和其源文件(TTY 类)有了一定的感性认识，下面来看看实际的命令行参数及 JDB 所支持的命令。

51.2.2 JDB 命令行

为了以最简单的形式启动 JDB，可在命令提示符下输入 jdb < 回车 > 。JDB 便开始执行其初始化并显示出一个 > 提示符。事实上，共有三种“正式的”命令行以各种模式启动 JDB，如下所示：

1. 启动 JDB 并在本机创建一个 VM 实例，同时不指定加载类：

```
JDB [ - dbgtrace] [ < java - args > ]
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

选　项	含　义
- dbgtrace	如果指定，允许从为了运行目标 Java 应用程序或小程序而创建的 JVM 实例中返回冗长的消息。这些消息被发送到 printToConsole 回调方法，具有[debugger: < 消息内容 >] 这样的格式。这样，我们就可以在 printToConsole 实现中过滤这些消息，并将其发送到日志文件窗口等。
< java - args >	这是一个可选的参数子集，我们可以在运行 java 命令以启动一个 JVM 实例时指定该参数。当前可识别的选项有：- cs、- checksource、- noasyncgc、- prof、- v、- verbose、- verify、- noverify、- verifyremote、- verbosegc、- ms、- mx、- ss、- oss、- D 以及 - classpath。

2. 启动 JDB 并在本机创建一个 VM 实例，以指定 < classname > 为装入类：

JDB [- dbgtrace] [< java - args >] < classname > [< args >]

选　项	含　义
- dbgtrace	同上。
< java - args >	同上。
< classname >	这一强制性参数是一个 .class 文件，该文件被初始加载到 JVM，并由调试器来控制。使用 run 命令可以开始执行该文件。
< args >	这一参数表示 < classname > 所需的参数。必须在此处指定，因为 JDB 启动后是没有办法为 < classname > 的 main 方法指定参数的。

3. 启动 JDB 并连接到一个已经在运行一个类的远程 VM 实例：

JDB [- dbgtrace] [- host < hostname >] - password < password >

选　项	含　义
- dbgtrace	同上。
- host < hostname >	这一可选参数指定一个计算机的 DNS 名或 IP 地址，该计算机将为所要调试的 Java 应用程序或小程序运行主机 JVM。如果未指定该参数，则将自动假设为 localhost(本地主机)。
- password < password >	该指示性参数是一个口令，当主机 JVM 将要调试的应用程序或小程序被装入时显示于控制台上。当相关的命令行标志设置时由 java 或 appletviewer 产生这一提示。

当我们输入 JDB 的一个命令行之后，前面描述的初始化过程便开始了，并最终出现交互提示符(>)。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

51.2.3 JDB 输入文件

如果我们选择使用一个特殊的命令集重复地运行一个调试过程,可以为 JDB 创建一个可以自动使用的命令输入文件。命令文件是一个简单的 ASCII 文件,其中的每一行包含一个有效的 JDB 命令,后面跟的是与运行 JDB 的操作系统(OS)所对应的行分界符(例如,对 Wintel 为 CR/LF)。对于一个命令输入文件,JDB 以表 51.19 中所列出的顺序查找三个地方。

表 51.19 JDB 命令输入文件的位置

目录(系统属性)	文件名	例 子
USER.HOME	JDB.INI	C:\JAVA\JDB.INI
USER.HOME	JDBRC	/JAVA/.JDBRC
USER.DIR	STARTUP.JDB	./STARTUP.JDB

如果找到前面提到的一个文件,JDB 将读入每一行并如同在控制台输入一样进行处理。如果我们需要 JDB 在处理完该文件后自动地退出,可在该命令文件的最后一行使用一个退出命令(quit 或 exit)。否则我们将留在 JDB 提示符下并保持这些处理命令的输出在控制台窗口是可见的。另外,由于对 printToConsole 的调用发送到了 System.out,所以我们可以使用命令行重定向指令,把这些命令的结果重定向到一个输出文件。

51.2.4 JDB 命令集

现在,我们已经知道了如何启动 JDB,关于如何操作 JDB 同样也很有用。为了说明 JDB 命令集的某些特性,我们使用了一个很小的带有线程的应用程序,其中的每个线程增加了一个共享的计数器,并显示计数器的值。程序清单 51.15 给出了这个应用程序。

```
public class MTTest extends Thread {
    static int count = 0;
    static Object sema = new Object();

    public MTTest( ThreadGroup theGroup, String threadName ) {
        super( theGroup, threadName );
    }

    public void run() {
        String myName = getName();
        while (true) {
            synchronized (sema) {
                if(count < 30) {
                    System.out.println( myName + ":" + count );
                    count += 1;
                } else break;
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                }
                yield();
            }
            System.out.println("Exiting" + getName());
        }

        public static void main( String[] args ) {
            ThreadGroup     theGroup;       // To contain the threads
            MTTest[]        theThreads;     // Array of threads
            // Wait for the user to hit <Enter> to start
            System.out.print( "MTTest: Press the <Enter> key to begin.");
            System.out.flush();
            try { System.in.read(); }
            catch (java.io.IOException e) {}
            System.out.println(" ");
            // Create the thread group
            theGroup = new ThreadGroup( "MTThreads" );
            // Create the thread array
            theThreads = new MTTest[3];
            // Create and start the threads
            for (int i = 0; i < theThreads.length ; ++i) {
                theThreads[i] = new MTTest( theGroup, "T" + Integer.toString(i));
                if (theThreads[i] != null)
                    theThreads[i].start();
            }
        }
    }
```

使用多线程化的 JDB 调试程序时首先需要知道的事情之一是，最好让目标应用程序等待，直到准备好之后在一个单独的处理空间中运行它。在一个单独的处理空间中运行可以避免目标应用程序的输出与调试器的输入交织在一起。另外，使应用程序等待直到我们准备好指的是它不会在启动用来运行该应用程序的 JVM 时就开始运行。

为了学习与每个命令相关的例子，我们完成以下步骤：

1. 使用下面的命令行对 MTTest 带有调试信息进行编译(调试信息包括行号和局部变量信息)：

```
java -g MTTest.java
```

2. 打开两个命令窗口：一个用于 JVM 运行 MTTest，另一个用于运行 JDB。从第一个命令窗口中，以如下方式启动 JVM：

```
java -debug MTTest
```

其中的 -debug 选项指示 JVM，我们将通过外部代理与它进行通信，MTTest 是将要加载

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的类文件的名字。当 java 启动以后，将在控制台显示出如下的信息：

```
Agent password = XXXXXX
```

此处的 XXXXXX 是下一步启动 JDB 时将要使用的口令。

3. 在第二个命令窗口中输入下述命令启动 JDB，并连接到正在运行的 JVM：

```
jdb -host localhost -password XXXXXX
```

4. 为了将调试会话置于感兴趣的点上，在 JDB 命令窗口中输入如下所示的命令行：

```
> stop in MTTest.run
Breakpoint set in MTTest.run
```

（不要担心命令的含义——我们将在后面的部分介绍它们。）

5. 在实际运行 MTTest(见第 2 步)的控制台窗口中，按回车键开始执行 MTTest。

我们会很快遇到断点并在 JDB 控制台窗口内看到如下内容：

```
Breakpoint hit: MTTest.run(MTTest:12)
T0[1]
```

现在，一切都已准备就绪，可以检查 JDB 所实现的特殊命令了。为了清楚起见，我们按照命令的功能将这些命令分成不同的组。通过使用这些分类，我们可以将每个命令归置到各自的分类中(见表 51.20)。

表 51.20 分类后的 JDB 命令

一般	上下关系	信息	断点	异常	线程
help/?	load	classes	stop	catch	suspend
exit/quit	run	dump	clear	ignore	resume
memory	threadgroup	list	step	kill	down
gc	thread	locals	next	up	
itrace	use	methods	cont		
trace			print		
!!			threadgroups		
			threads		
			where		

注意：kill、next、itrace 及 trace 是文档中未收录但却实现了的命令。

本节中的以下部分介绍每个命令及其功能。

51.2.5 一般命令

这些命令用于控制调试器的某些特性或查询远程 JVM 的状态。

help/? 语法：help[或?]

该命令显示出 JDB 所支持的文档中包含的命令列表。

exit/qiut 语法：exit[或 qiut]

使用：RemoteDebugger.close( )

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

该命令结束我们的调试对话及 JDB。JDB 与远程 JVM 之间的连接断开，如果本地调试，则 VM 关闭。

memory　语法：memory

使用：RemoteDebugger.freeMemory()和 RemoteDebugger.totalMemory()

该命令显示远程 JVM 中内存总数及可使用的内存数。例如，在笔者的系统中 memory 命令将显示如下信息：

```
Free:2674104, total:3145720
```

gc　语法：gc

使用：RemoteDebugger.gc( )

该命令导致垃圾收集任务在远程 JVM 上运行，调试器已经不使用了的那些类将被释放。JDB 自动通知 JVM 不对包括在当前 RemoteThreadGroup 和 RemoteThread 实例中的类进行垃圾收集。如果我们有一个很长的调试对话过程，则应不时地使用 gc 命令清除那些已经由远程 JVM 为我们缓存起来的 RemoteClass 实例。

itrace(该命令为一个非文档化的命令)　语法：itrace on | off

使用：RemoteDebugger.itrace( )

该命令打开(on)或关闭(off)正在主持应用程序的远程 JVM 的字节码指令跟综功能。其输出发送到远程 JVM 的 System.out，并且不能在调试对话中截取这一输出。

下面是 itrace 的一个样本输出：

```
1393B58 6B4AD8 ifeq goto 6B4ADD(taken)
1393B58 6B4ADD aload_0 = > java.net.SocketInputStream@139EE80/1481298
1393B58 6B4ADE aload_1 = > byte[][2048]
1393B58 6B4ADF iload_2 = > 0
1393B58 6B4AE0 iload_3 = > 2048
1393B58 6B4AE1 invokenonvirtual_quick
java/net/SocketInputStream.socketRead([BII)I (4)
```

trace(该命令为一个非文档化的命令)　语法：trace on | off

使用：RemoteDebugger.trace( )

该命令打开(on)或关闭(off)正在主持应用程序的远程 JVM 的方法跟踪功能。输出发送到远程 JVM 的 System.out 并且不能在调试对话中进行截取。

下面是 trace 命令的一个输出样本：

```
# Debugger agent [ 3] ¦ ¦ ¦ < java/lang/Runtime.traceMethodCalls(Z)V returning
# Debugger agent [ 2] ¦ ¦ < sun/tools/debug/Agent.handle(ILjava/io/ DataInputStream;
Ljava/io/DataOutputStream;)V returning
# Debugger agent [ 2] ¦ ¦ > java/io/DataOutputStream.flush()V (1) entered
# Debugger agent [ 3] ¦ ¦ ¦ > java/io/BufferedOutputStream.flush()V (1) entered
# cbugger agent  [ 4] ¦ ¦ ¦ ¦ > java/net/SocketOutputStream.write([BII)V (4) entered
# Debugger agent [ 5] ¦ ¦ ¦ ¦ ¦ > java/net/SocketOutputStream.socketWrite([BII)V (4) entered
# Debugger agent [ 5] ¦ ¦ ¦ ¦ ¦ < java/net/SocketOutputStream.socketWrite([BII)V returning
# Debugger agent [ 4] ¦ ¦ ¦ ¦ < java/net/SocketOutputStream. Write([BII)V returning
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
# Debugger agent [ 4] ¦ ¦ ¦ ¦ > java/io/OutputStream.flush()V (1) entered
# Debugger agent [ 4] ¦ ¦ ¦ ¦ < java/io/OutputStream.flush()V returning
# Debugger agent [ 3] ¦ ¦ ¦ < java/io/BufferedOutputStream.flush()V returning
# Debugger agent [ 2] ¦ ¦ < java/ioDataOutputStream.flush()V returning
# Debugger agent [ 2] ¦ ¦ > java/io/FilterInputStream.read()I (1) entered
# Debugger agent [ 3] ¦ ¦ ¦ > java/io/BufferedInputStream.read()I (1) entered
# Debugger agent [ 4] ¦ ¦ ¦ ¦ > java/io/BufferedInputStream.fill()V (1) entered
# Debugger agent [ 5] ¦ ¦ ¦ ¦ ¦ > java/net/SocketInputStream.read([BII)I (4) entered
# Debugger agent [ 6] ¦ ¦ ¦ ¦ ¦ ¦ > java/net/SocketInputStream.socketRead([BII)I (4) entered
```

调用信息的格式使用了在类文件部分中描述的签名，它是在.class 文件结构的后续部分描述的。

!!（**重复上一命令**） 语法：!!

该命令重新执行或重复上一输入命令。它不是由远程类实现的，它只是 JDB 命令处理器所允许的一个特性。

51.2.6 上下关系命令

这些命令用于建立调试对话的上下文关系。它们通过 TTY 操作设置远程 JVM 的状态和实例变量。为了使用 JDB 中的几乎所有命令，必须设置当前线程组和当前线程。初始的上下文关系在使用 run 命令时自动设置，否则，必须使用 threadgroup 和 thread 命令自己设置。

load 语法：load <classname>

使用：RemoteDebugger.findClass( )

该命令导致远程 JVM 搜索并装入指定的 <classname>。如果我们没有完整地给出待装入类的合法名称，VM 将试着查找常用的包补齐该名字。如果没有找到指定的类，返回一个错误消息。另外，如果未指定 <classname> 也将显示一个错误消息。该命令不影响当前的上下文关系。

run 语法：run [<classname> [args]]

使用：RemoteClass.getQualifiedName 和 RemoteDebugger.run( )

该命令装入并开始执行指定的 <classname> 或由上次调用 run 命令时指定的 <classname>。如果指定类没找到或启动 <classname> 时出现一般性故障，则返回错误消息。该命令也可以通过为 currentThreadGroup 和 currentThread 建立初始值设置上下文关系。

threadgroup 语法：threadgroup <thread group name>

使用：RemoteDebugger.listThreadGroup 和 RemoteThreadGroup.getName

该命令通过把对 RemoteThreadGroup 实例的引用保存在 currentThreadGroup 实例变量中，建立指定的 <thread group name>（<线程组名>）为缺省线程组。在使用任何有关断点、异常和线程管理的命令时需要该命令。例如，我们可以输入：

```
> threadgroup MTThreads
```

来指定当前缺省的线程组。

thread 语法：thread t@ <thread id> | <thread id>

<thread id> 是一个表示线程 ID 号的整型常量（见 threads 命令）。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用:RemoteThreadGroup.listThreads

该命令通过把对其 RemoteThread 实例的一个引用保存在 currentThread 实例变量中,设置指定的 < thread id > 为当前线程组的上下文关系中的当前线程。在使用任何与断点、异常及线程管理有关的命令时需要该命令。例如,我们可以输入:

```
> thread 5
T0[1]
```

来指定当前缺省的线程。T0[1] 是现在显示给我们的新的提示,说明我们的上下文关系处在线程 T0 中,它是当前线程组中的第一个线程。

use　语法:use [source file path(源文件路径)]

使用:RemoteDebugger.getSourcePath 和 RemoteDebugger.setSourcePath

该命令用于显示或设置路径,远程 JVM 使用该路径查找.class 和.java 文件。如果不带参数调用该命令,则显示出当前源文件路径。如果使用一个指定路径调用该命令(与类路径系统属性的格式相同),则按照指定路径更新源文件路径。例如,为了显示当前的类或源文件路径并修改该路径,可使用如下方式:

```
> use
.;c: \ java \ lib \ classes.zip
> use .;c: \ java \ lib \ classes.zip;c: \ java \ lib \ classdmp.zip
> use
.;c: \ java \ lib \ classes.zip;c: \ java \ lib \ classdmp.zip
```

51.2.7　信息命令

这些命令用来显示关于当前加载到远程 JVM 并被远程 JVM 所知的类的信息。这些命令本质上是面向列表的,依赖于前面介绍过的当前建立的上下文关系。

classes　语法:classes

使用:RemoteDebugger.listClasses 和 RemoteClass.description

该命令显示为当前正在主持调试目标的远程 JVM 所知的类和接口的名字。如果这一列表非常地大,试使用 gc 命令释放那些由远程 JVM 和 RemoteDebugger 代理为我们保留的 RemoteClass 实例。

下述内容是启动 MTTest 后,classes 命令的输出样本:

```
0x1393768:class(MTTest)
0x1393778:class(sun.tools.debug.Agent)
0x13937a0:class(java.lang.Runtime)
0x1393818:class(java.net.ServerSocket)
0x1393830:class(java.net.PlainSocketImpl)
0x1393840:class(java.net.SocketImpl)
0x1393890:class(java.net.InetAddress)
```

dump　语法:dump t@ < thread id > | $ s < slot id > |0x < class id > | < name >

t@ < thread id > 表示当前线程组中一个有效的线程 ID;$ s < slot id > 表示堆栈帧中一个变量的插槽/偏移量;0x < class id > 表示当前所加载的类的数值标识符;< name > 表示常量

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

this、一个合法类名、一个字段名(例如,class.field)、一个参数名或一个局部变量名。

使用: RemoteThreadGroup. listThreads、RemoteThread. getStackVariables, RemoteStackVariable.getValue、RemoteDebugger.get 和 RemoteDebugger.findClass

该命令卸出指定线程、基于堆栈的变量、类、字段、命名的局部变量或命名的参数的详细描述。如果请求一个参数、变量或字段,则显示其名字和值。如果指定的是一个线程或类,则显示出描述线程或类的详细信息,包括实例变量和其当前值。

下面是一个卸出 MTTest 类的例子:

```
T0[1] dump MTTest //也可以输入:dump 0x13933768
"MTTest" is not a valid field of (MTTest) 0x13a0ca8
MTTest = 0x1393768:class(MTTest){
superclass = 0x1393008:class(java.lang.Thread)
loader = null
private static Thread activeThreadQ = null
private static int threadInitNumber = 2
public static final int MIN_PRIORITY = 1
public static final int NORM_PRIORITY = 5
public static final int MAX_PRIORITY = 10
static int count = 0
}
T0[1]
```

注意第二行是 dump 命令使用搜索算法的结果。

list 语法:list [line number(行号)]

使用: RemoteThread. getCurrentFrame、StackFrame. getRemoteClass、RemoteClass. getSourceFileName、RemoteClass.getGetSourceFile

该命令显示当前线程的当前方法的一行或多行源代码。在上下文关系中必须有一个线程正在运行,但处于挂起状态。另外,如果指定行号的话,必须是相对于定义当前方法的源文件顶部的。否则,如果不指定行号,则显示当前行。所显示的源代码行列表包括指定行前后各四行。

下面是不使用参数在当前断点查看 MTTest 的情况:

```
T0[1] list
6       }
7
8       public void run( ){
9
10 = >              String myName = getName( );
11
12                  while (true){
13                      synchronized(sema){
14                          if(count < 30){
T0[1]
```

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友, Java群空间; 可以联系群主获取更多大型企业内部技术教程。

第 10 行的 = > 表示当前源代码行。

locals　语法:locals

使用:RemoteThread.getStackVaribles

该命令显示当前方法的所有参数和本堆栈帧中定义的变量。在上下文关系中必须有一个线程,并且必须使用 - g 选项编译代码,以便获得调试可用的局部变量和参数的符号表信息。

例如,如果输入 locals,则我们将看到如下结果:

```
T0[1]locals
Method arguments:
this = Thread[T0,5,MTThreads]
Local variables:
myName is not in scope.
T0[1]
```

this 参数在所有方法中都给出并且由 JVM 调用逻辑隐式地压入堆栈中。myName 变量还没有进入其作用范围,我们正处在方法开始处的断点上。

methods　语法:methods < classname > | 0x < class id >

使用:RemoteDebugger.get 或 RemoteDebugger. findClass 和 RemoteClass.getMethods

该命令显示指定类中的所有方法,包括每个方法的签名。

对 MTTest 类所列出的方法如下所示:

```
T0[1]methods MTTest
void <init>(ThreadGroup,String)
void run( )
void main(String[])
T0[1]
```

< init > 方法是一个特殊的名字,表示本类的构造器。

print　语法:print t@ < thread id > | $ s < slot id > | 0x < class id > | < name >

t@ < thread id > 表示当前线程组内的一个有效线程 ID, $ s < slot id > 表示一个堆栈帧中的一个变量的插槽/偏移量;0x < class id > 表示当前所加载类的数值标识符; < name > 表示常量 this、有效类名、字段名(例如,class.field)、参数名或局部变量名。

使用: RemoteThreadGroup. listThreads、RemoteThread. getStackVariables、RemoteStackVariable.getValue、RemoteDebugger.get 以及 RemoteDebugger.findClass

该命令显示指定线程、基于堆栈的变量、类字段、命名的局部变量或命名的参数的一个简单描述。如果请求一个参数、变量或字段,则显示其名字和值;如果指定的是一个线程或类,则显示指定线程或类的名字和 ID。

以下是打印 MTTest 类的一个例子:

```
T0[1] print MTTest
MTTest = 0x1393768:class(MTTest)
T0[1]
```

threadgroups　语法:threadgroups

Java学习群: 72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用：RemoteDebugger. listThreadGroups、RemoteThreadGroup. getName 以及 RemoteThreadGroup. description

该命令显示远程 JVM 中所有活动线程的名字和描述信息。

对 MTTest 使用 threadgroups 命令的结果如下所示：

```
T0[1]threadgroups
1. (java.lang.ThreadGroup)0x13930b8 system
2. (java.lang.ThreadGroup)0x1393ec60 main
3. (java.lang.ThreadGroup)0x1393a0600 MTThreads
T0[1]
```

threads 语法：threads [thread group name(线程组名)]

使用：RemoteDebugger. listThreadGroups、RemoteThreadGroup. getName、RemoteThreadGroup. listThreads 、RemoteThread. getName 、RemoteThread. description 以及 RemoteThread. getStatus

该命令显示当前或指定线程组的线程列表。如果当前或指定线程组含有嵌入的线程组，则嵌入线程组中的线程也列出。

对 MTTest 的名为 MTThreads 的线程使用 threads 命令，则应显示与下述内容类似的信息：

```
T0[1]threads MTThreads
Group MTThreads:
1. (MTTest)0x13a0b30 T0 at breakpoint
2. (MTTest)0x13a0b90 T1 suspended
3. (MTTest)0x13a0bd0 T2 suspended
T0[1]
```

where 语法：where [all| < thread id >]

使用：RemoteThreadGroup. listThreads、RemoteThread. dumpStack 以及 RemoteStackFrame. toString

该命令显示当前线程(使用 thread 命令设定)的调用堆栈(为了得到该点所调用的方法列表)、所有线程(使用 threadgroup 命令设置的当前线程组中)、或者指定的线程(通过其 ID)。

在笔者的系统中，where all 命令的显示内容如下所示：

```
T0[1] where all
Finalizer thread:
Thread is not running(no stack).
Debugger agent:
[1]sun.tools.debug.Agent.handle (Agent:590)
[2]sun.tools.debug.Agent.run (Agent:324)
[3]java.lang.Thread.run (Thread:294)
Breakpoint handler:
[1]java.lang.Object.wait (Object:152)
[2]sun.tools.debug.BreakpointQueue.nextEvent (BreakpointQueue:51)
[3]sun.tools.debug.BreakpointHandler.run (BreakpointHandler:184)
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
main:
[1] MTTest.main (MTTest:51)
T0:
[1] MTTest.run (MTTest:12)
T1:
Thread is not running(no stack).
T2:
Thread is not running(no stack).
T0:[1]
```

51.2.8　断点命令

这些命令允许我们设置/删除断点,并利用断点控制执行流程。断点是大多数调试过程都要使用的东西,这也大概是调试时停止应用程序以便做一些其它事件的唯一手段。或者说在使用 Java 的情况下,一个线程必须在某些点上无条件停止下来,这就是断点的作用。当我们建立了一个断点之后(我们已经在设置调试对话的第 4 步中看到了有关这一点的简要介绍),可以执行程序/线程直到执行路径遇到该断点。当遇到断点时,线程的执行停止,我们重新获得了对远程 JVM 和应用程序的控制。

现在,我们可以设置和删除断点,并使用 step 和 next 命令"遍历"自己的程序,或使用 cont 命令继续执行程序。

stop　语法 1:stop in <classname>.method|0x<class id>.method

使用:RemoteDebugger.findClass 或 RemoteDebugger.get 、RemoteClass.getMethod、以及 RemoteDebugger.setBreakpointMethod

语法 2:stop at <classname>:line number|0x<class id>:line number

使用:RemoteDebugger.findClass 或 RemoteDebugger.get 和 RemoteClass.setBreakpointLine

该命令在指定方法(语法 1)的第一个字节码指令处,或在指定行的第一个字节码指令处设置断点。如果使用语法 2,行号是相对于包含指定的 <classname> 或 <class id> 的源文件起始处的。如果不带参数执行 stop 命令,则显示现有的断点。当一个断点设置在作为一个多线程应用程序或小程序一部分的一个方法上时,则当穿越该方法或代码行时,断点将作用于所有活动的线程。在使用 clear 命令删除一个断点之前,它一直处于活动状态。

下面的例子列出了当前的断点,之后在 MTTest 的第 12 行上设置一个断点,最后再次显示断点列表:

```
T0[1] stop
Current breakpoints set:
MTTest:10
T0[1] stop at MTTest:12
Breakpoint set at MTTest:12
T0[1] stop
Current breakpoints set:
MTTest:12
MTTest:10
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
T0[1]
```

clear 语法 1:clear <classname>.method|0x<class id>.method

使用:RemoteDebugger.findClass 或 RemoteDebugger.get、RemoteClass.getMethod 以及 RemoteClass.clearBreakpointMethod

语法 2:clear <classname>:line number|0x<class id>:line number

使用:RemoteDebugger.findClass 或 RemoteDebugger.get 和 RemoteClass.clearBreakpointLine

该命令清除指定方法(语法 1)的第一条字节码指令或指定行的第一条字节码指令上已有的断点。如果使用语法 2,行号相对于包含 <classname> 或 <class id> 的源文件的起始处。如果不带参数使用 clear 命令,则显示现有的断点。当从作为一个多线程应用程序或小程序一部分的一个方法清除一个断点时,则当穿越该方法或代码行时,将影响到所有活动线程。

下面的例子首先列出当前的所有断点,然后清除 MTTest.run 起始处的一个断点,最后再次显示所有断点:

```
T0[1] clear
Current breakpionts set:
MTTest:12
MTTest:10
T0[1] clear MTTest.run
Breakpoint cleared at MTTest.run
T0[1] clear
Current breakpoints set:
MTTest:12
T0[1]
```

step 语法:step

使用:RemoteThread.step

该命令执行当前停止线程的下一条指令。如果下一条指令是一个方法调用,执行停止在正在被激活的方法的第一条指令上。如果不存在当前线程或当前线程并未挂起在一个断点上,则产生一个错误。

next(该命令为一个未文档化的命令) 语法:next

使用:RemoteThread.next

与 step 命令一样,next 命令单步执行当前停止的线程到下一条指令。但是,如果下一条指令是一个方法调用,则调用该方法并根据正在被执行的方法的返回情况返回调试器。在这一点上,当前的指令是紧跟在该调用之后的指令。与 step 命令一样,如果没有当前线程或当前线程并未挂起在一个断点上,则产生一个错误。

cont 语法:cont

使用:RemoteThreadGroup.listThreads、RemoteThread.cont 以及 RemoteThread.resetCurrentFrameIndex

该命令继续缺省线程组中所有挂起线程的执行。当我们已经完成了一个线程的单步调试,并且想使应用程序直接执行,直到下一个断点或发生异常时,使用该命令。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

51.2.9　异常命令

这些命令控制 JDB 应捕获或忽略的异常类。Java 的一个更有意思的方面是它的异常概念。异常好像是智能断点，我们可以直接在自己的应用程序中编写处理异常的逻辑代码。异常一般用来捕获非常特殊的异常情况，这种情况在通常的情况下可能并不发生。

JDB API 的一个特性是，它可以注册我们所感兴趣的特殊异常，并使其行为类似于一个断点，以便可以步入到由 catch 块中异常处理代码编写的逻辑代码中。如果我们不选择以这种方式处理断点，那么调试客户会接到一个通知，如同遇到了一个不可恢复的断点。控制返回到调试器，但我们将无法步入到 catch 块中的异常处理逻辑。

catch　语法：catch [< classname > 0x < class id >]

使用：RemoteDebugger. getExceptionCatchList、RemoteDebugger. findClass 或 RemoteDebugger. get 以及 RemoteClass. catchException

该命令导致调试器捕获（通过 DebuggerCallback. exceptionEvent）远程 JVM 所抛出的由 < classname > 或 0x < class id > 指定的 exception 类异常。抛出 exception 类异常可导致执行停止，就好像在处理指定的异常时当前活动的 catch 块中的第一条可执行语句上设置了一个断点一样。换句话说，应用程序中“捕获”指定异常的所有 try - catch 块都将成为一个断点。如果未指定类，则显示已经捕获到的异常。如果指定的类不是 Exception 类的子类，则产生一个错误。

ignore　语法：ignore [< classname > 0x < class id >]

使用：RemoteDebugger. getExceptionCatchList、RemoteDebugger. findClass 或 RemoteDebugger. get 以及 RemoteClass. ignoreException

该命令导致调试器停止捕获远程 JVM 所抛出的由 < classname > 或 0x < class id > 指定的 exception 异常。这并不停止 exception 异常的抛出，但它确实停止了调试器对 exception 异常的捕获，如同它是一个断点一样。如果未指定类，则显示已经捕获到的异常。如果指定类不是 Exception 的一个子类，则产生一个错误。

51.2.10　线程命令

这些命令用来控制当前活动线程的执行状态及堆栈。线程控制命令与手工断点有些相似，使用它我们可以在其跟踪过程中停止一个线程而无需设置断点。当无意中编写了一个死循环并且看不出毛病存在于何处时，这些命令会非常有用。我们也可以对已经挂起的线程恢复执行，也可以完全清除它。当我们挂起一个线程之后，可以对处于活动状态的调用堆栈中的当前帧进行操作。这将允许我们对调用当前挂起方法的那些方法的参数及局部变量进行检查。

suspend　语法：suspend [thread - id [thread - id...]]

使用：RemoteThreadGroup. listThreads 和 RemoteThread. suspend

该命令挂起（停止）指定的一个或多个线程的执行，如果未指定线程，则挂起所有非系统线程的执行。这将导致在所涉及到的线程中，当前正在执行的指令上发生一个一次性断点。

resume　语法：resume [thread - id [thread - id...]]

使用：RemoteThreadGroup. listThreads 和 RemoteThread. resume

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

该命令恢复（继续）指定的一个或多个线程的执行，如果未指定线程，则恢复所有非系统线程的执行。这将允许所涉及到的线程好像在先前挂起的代码处从没有存在过断点一样继续执行。

kill（该命令为一个未文档化的命令） 语法：kill < thread group name（线程组名）< thread id >

使用：RemoteThreadGroup. listThreads 、RemoteThread. stop 以及 RmoteThreadCroup. stop

该命令结束（永久性停止）指定线程组中所有线程的执行，或者只结束指定线程（使用ID）的执行。当结束一个线程或线程组之后，不能再对其进行恢复，并且也不能再使用任何断点命令（step、next 及 cont）。如果没有指定参数或者所指定的名字或 ID 无效，则产生一个错误。

up 语法：up [n frames]

使用：RemoteThread. up

该命令将上下文关系中的堆栈帧从当前位置向上移动一帧（缺省情况）或 n 帧。一帧表示正在执行的一个方法在其调用并进入另一方法之前的执行状态。执行状态包括方法调用时正在执行的行位置、该行的参数（及其当前值）、以及该行的局部变量（及其值）。在这一点上，该方法的状态被“压栈”并且为正在被调用的方法创建了一个新的帧。当每次从一个方法调用另一个方法时，当前帧就被“压栈”并将一个新帧置于上下文关系中。通过向“上”移动调用栈，可以检查前面方法的参数和变量。

down 语法：down [n frames]

使用：RemoteThread. down

该命令在使用 up 命令之后使用。down 命令将上下文关系中的当前堆栈帧从其当前位置向下移动一帧（缺省情况）或 n 帧。通过向“下”移动调用栈，我们可以先于当前线程中最后的挂起点（不管是一个断点、异常捕获还是显式调用 suspend ）到达执行的当前状态。

51.2.11 JDB 总结

从命令的个数（33 条不同的命令）上我们可以看出，JDB 实际上是一个功能非常强大的调试工具。

它是如何使用 Sun 公司的 JDB API 实现一个调试器的一个很好的例子。另外，它也是 Java 系统编程的一个很好的样本应用程序。实际上给 TTY 类包装一些 GUI 功能使它在视觉上更具吸引力并不需做太多的工作（建立子类、为输出增加一个多行文本字段、为命令组设置菜单项、创建弹出式窗口等等）。即使我们不以这种方式使用 GUI 功能，也可以在 Sun 公司的 JDK 所支持的任何平台上，使用 JDB 调试自己的应用程序。

调试策略又怎样呢？应当记住的最重要的一点是，如果我们在一个多线程的应用程序中遇到了问题，可以充分地使用断点和异常。在我们自己的代码中抛出异常的好处是，可以像断点一样捕获它们，而不需知道特定的源代码行号或方法名称，也可以忽略它们。（切记，一个异常并不一定意味着发生了灾难性的错误，它只是一段代码给另一段代码的一个信号。）我们可以创建一个异常，当一个循环计数器超过了一个特定的值时，当一个套接字收到一条特殊消息时，或者其它情况发生时将其抛出。异常的使用方式实际上没有任何限制。

另一个可以使用的重要特性是 where 命令。我们不能认识到自己正处于一个递归性的状

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

况下，除非查看调用堆栈的情况。我们可能是在有意识地进行递归调用，但在一般情况下这并不是普通商业应用程序所使用的技术。当我们感觉到需要查看调用堆栈时，可以使用 up 和 down 命令，在调用堆栈中移动上下文，并检查可能影响程序执行的局部变量或参数。

最后一个策略是彻底理解编译后的类文件模式以及 JVM 使用它们的方式。这是下面两章将要介绍的内容。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 52 章　理解.class 文件

本章内容：

52.1　基本度量
52.2　.class 文件的元素
52.3　定义
52.4　.class 文件的结构
52.5　现在可以做些什么

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

52.1　基本度量

.class 文件是 Java 虚拟机(Java)应用程序的基本度量单位。它代表了编译器与 JVM 实现之间的分类约定。我们在这里使用“编译器”而不使用“Java 编译器”是因为,任何语言的编译器都具有潜在地生成.class 文件和 Java 字节码的能力,我们随后就会看到这一点。

实际上,.class 文件是表示绝对动态的结构和数组的一个有序的字符集,它描述了可执行单元的编译版本(或运行图),称之为类。组成.class 文件的大部分组件都有一个固定的结构,其后是一组可变长的结构。有些是强制性的,有些是可选的。应注意的重要的一点是,产生一个.class 文件的过程必须严格按照本章所讲的格式和风格进行。否则,JVM 的类加载器和检查器将不接受所提交的.class 文件。

52.2　.class 文件的元素

保持.class 文件为面向字节流的格式对于快速装入和分析其中所包含的信息是很关键的。类加载器的实现者可能利用了 Java 的流 I/O 类——例如,以方便地逐段读入一个.class 文件,并在读入时进行分析,或者将其读入一个字节数组中然后手工地进行分析。

必须牢记的一个概念是,我们必须顺序地读入每个部分,直到读完。另外,不首先读完其描述信息是不能真正读入文件的一个部分的。例如,文件中有一个称为常量池(Constant Pool)的部分,我们首先读入的项是随后的元素个数,然后,是每个要读入元素的描述符,描述符可告诉我们下一个元素的格式,最后,根据其特定的格式读入实际的元素。

一个文件可分为如下几个逻辑部分:

- 在最高层,.class 文件表示一个编译过的单个 Java 类。当我们编译一个 Java 源文件(.java 文件)时,对该文件中所定义的每一个类,编译器都将产生一个.class 文件。
- 第二层,为基本类结构,该层中包含了关于类的特性及用于描述常量值的部分。这些常量值是在编译时为类(常量池)所收集的,之后是类的接口、字段、方法以及类层属性。
- 最后一层,每个子部分又可看作是独立部分:常量池及其元素、接口表、包括特性和属性的字段表、以及包括特性和属性的方法表。

> **注意**:本章所描述的信息是从以下两个地方搜集而来的:
>
> - 原先的 JavaSoft Web 站点(www.javasoft.com)和 Sun 的 Web 站点(www.sun.com)的在线参考资料。
> - 1996 年 5 月 Sun 公司在旧金山召开的全球 Java 开发者大会(Worldwide Java Develeper Conference),JavaOne 分会的资料。

52.3　定义

为了全面理解.class 文件的内容,我们需要首先定义一些该文件所包含的各个部分要使用的公共结构。在这一节中,我们要检查常量池(Constant Pool)、签名格式或类型定义以及

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

属性。

52.3.1 常量池

常量池这一概念对我们来说也是一个新概念,但是在早期的编译器及运行的系统中,就已经使用常量池这一概念了。常量池中包含了编译一个类的源代码时所遇到的每个明显的常量值。此处,一个常量值可能是一个真正的数字值,也可能是一个字符串常量、类名、类型描述或方法签名。

当每次遇到这些常量值时,都要在常量池中搜索与之匹配的值,以避免重复存放该值。如果在常量池中找到了该值,则将其在常量池中的现有位置插入到类定义或编译过的字节码流中。如果在常量池中没有找到该值,则将其添加到常量池中。在加载类时,常量池被放置在内存中一个与数组类似的结构中以便快速访问。然后加载类的其余部分。在运行时如果需要某一常量,则首先使用其索引在常量池中定位该值,然后提取它。

常量池的使用使得编译后的类文件更小,所以加载速度也就更快。在运行时 Sun 公司的 JVM 实现采用了一种机制,这种机制使得只有在第一次需要一个明显的值时才确定一个常量池引用。在这之后,可以使用常量池之外的一个特殊数组直接引用所确定的值。这种机制实际上是由一组称为快速指令的内部字节码来支持的。由于它们是严格依赖于实现的,所以它们并不是正式 Java 字节码定义的一部分。

记录在.class 文件中的常量池采用了一种非常紧凑的格式。它使用一个 16 位无符号整数值作为开始,该值为其后的元素个数加一。(所增加的那个额外计数是指只在运行时使用的第零个元素,它不包括在.class 文件所包含的元素中。)在这个计数值之后是一个可变长数组,该数组的每个元素是一个可变长的结构,并且在每个元素之间不存在间隔。

有十二种类型的值和相关结构可存储在常量池中。每种结构使用一个单字节大小的称为标签的整型值开始(见表 52.1)。该标签用于确定在这之后组成这一元素的结构的其余部分的字节格式。

表 52.1 常量池标签

标签	含义	注释
1	Utf8 字符串	
2	统一字符串	此处未使用
3	Integer 值	
4	Float 值	
5	Long 值	
6	Double 值	
7	类引用	只引用类名
8	字符串	
9	字段引用	仅在字节码流中使用
10	方法引用	仅在字节码流中使用
11	接口方法	仅在字节码流中使用
12	名字和类型引用	

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

现在我们知道了标签值，让我们来看看每个常量池元素的类型。所有的标签长度都是一个字节，所有的长度和索引都是 16 位无符号整型值，除非有附加注释。

标签 1——Utf8 字符串　Utf8 常量用于以尽可能小的表示形式来表示统一码(Unicode)字符串值(见表 52.2)。在 Utf8 字符串中，一个字符将根据其值的不同占用 1 到 3 个字节。它主要是针对 ASCII 字符值的，可使所有非空的 ASCII 字符存放在一个单一字节中。.class 文件非常依赖于这一常量池入口类型，其中所有实际的字符串值(包括类、字段、方法名、类型及方法签名)都以 Utf8 常量形式存储。

表 52.2　Utf8 字符串常量

字段	字节数	值
Tag	1	1
Size	2	Utf8 字符串的字节长度
Data	Size	实际的 Utf8 字符串

标签 2——统一码字符串　统一码常量用来存放一个实际的统一码字符串，它不是使用于.class 文件本身(见表 52.3)。它可能被内部使用以便在运行时保存一个真实的统一码字符串。统一码常量的格式类似于 Utf8 常量，但每个字符都是一个真正的统一码字符。

表 52.3　统一码字符串常量

字段	字节数	值
Tag	1	1
Size	2	统一码字符串的字节长度
Data	(Size * 2)	实际的统一码字符串

标签 3 和 4——Integer 和 Float 值　Integer 和 Float 常量分别用于存放整型和浮点型常量值，它们可以作为字段或变量的初始值，也可以作为 Java 语句中的硬编码常量值(见表 52.4)。

表 52.4　Integer 和 Float 常量

字段	字节数	值
Tag	1	Integer 常量为 3、Float 常量为 4
Data	4	实际的整数或浮点数值，采用由大到小(最高有效位(MSB)在前)的顺序

标签 5 和 6——Long 和 Double 值　Long 和 Double 常量分别用来存放长型和双精度型常量，可作为字段或变量的初始化值，也可作为 Java 语句中的硬编码常量值(见表 52.5)。由于内部的原因，每个 Long 和 Double 常量使用了常量池中的两个元素。所以如果一个 Long 常量从常量池中的位置 4 开始，则下一个常量将从位置 6 开始。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 52.5 Long 和 Double 常量

字段	字节数	值
Tag	1	Long 常量为 5，Double 常量为 6
Data	8	实际的长型或双精度型值，采用由大到小（最高有效位（MSB）在前）的顺序

标签 7——类引用 类引用常量是一个间接值，用于指向一个类的实际常量名（见表 52.6）。.class 文件中使用的所有类名都是采用这种方式引用的，除非它使用在字段、变量、参数、或返回类型的声明中（见稍后的“类型信息”部分）。另外，由于 Java 中数组是对象，所以所有的数组引用都是以类引用常量为基础的。

表 52.6 类引用常量

字段	字节数	值
Tag	1	7
Index	2	常量池中一个 Utf8 字符串的位置，包含完整合法的类名字

标签 8——字符串引用 字符串引用也是一个间接值，当在类定义或字节码流中遇到实际的字符串常量时使用（见表 52.7）。这种字符串可以在一个 Java 表达式中作为一个字符串变量的初始化值，或在一个方法调用中作为一个参数直接使用。

表 52.7 字符串引用常量

字段	字节数	值
Tag	1	8
Index	2	常量池中一个 Utf8 字符串的位置，包含实际的字符串值

标签 9、10 及 11——字段、方法及接口方法引用 字段、方法及接口方法引用常量用于编译过的 Java 字节码流中，以便动态地引用驻留在另一个类或接口中的一个字段或方法（见表 52.8）。类引用用于动态地加载所引用的类，名字和类型引用用于找出指定的字段或方法，以便使用或调用。

表 52.8 字段、方法及接口方法引用常量

字段	字节数	值
Tag	1	字段为 9、方法为 10、接口方法为 11
Class Index	2	常量池中一个类引用的位置，包含其后的字段或方法引用
Name/Type	2	常量池中一个名字和类型索引引用的位置，描述一个字段或方法

标签 12——名字和类型引用 名字和类型引用用来存放字段、方法或参数的实际名字及其相关的类型或签名（见表 52.9）。这些常量类型用于任何定义或使用字段、变量、方法或参数的地方。Description（描述）字段内容的确切格式，见下一节“类型信息”。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 52.9　名字和类型引用常量

字段	字节数	值
Tag	1	12
Name Index	2	常量池中一个 Utf8 字符串的位置，包含字段、变量、参数或方法的名字
Description Index	2	常量池中一个 Utf8 字符串的位置，包含名字的类型或签名

52.3.2　类型信息

为了有一个一致的方法来描述字段、变量、参数及方法签名的数据类型，.class 文件使用了一个非常简短的注释。基本上说，对 JVM 可识别的每一种本地类型使用了其全名的一个单字符缩写来表示，对于类和数组使用了一个特殊的修饰字符来表示。在常量池中每个类型和签名的缩写采用一个 Utf8 字符串格式保存。对字段或变量的类型，只是一个单一的类型描述；对方法签名，则是一系列的类型描述，首先是各参数的类型（按顺序并且用括号括起），然后是方法的返回类型的缩写。

表 52.10 给出了缩写类型名及其实际数据类型。

表 52.10　.class 文件中所使用的数据类型缩写

缩写	Java 类型	注释
B	byte	
C	char	
D	double	
F	float	
I	int	
J	long	
S	short	
Z	boolean	
V	void	
L<类名>;	类	大写字母后面是完整合法的类名，以分号结束。注意使用正斜杠而不是句点来分隔实际标记类名的包名
[	数组维	一个单一的方括号用于表示一个数组的每一维

为了看到这些缩写是如何使用的，我们来看看程序清单 52.1。在此我们定义了一个简单的 Java 类，并且为每一个变量和方法在注释中给出其缩写。

```
class foo {

//    TYPE            FIELD NAME              SHORT-HAND VERSION
      int             simpleInt;              // I
      boolean         simpleBool;             // Z
      float[]         floatArray;             // [F
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
char[][]          twoDimCharArray;          // [[C
String[][][]      threeDimStringArray;      // [[[Ljava/lang/String;
// Note the use of slashes here

void DoSomething( long arg1, double[][] arg2 ) { }
// (J[[D)V
// Two arguments, a long and a two dimension double array, returning
// nothing.

java.net.Socket OpenSocket( String hostname, int port ) { }
// (Ljava/lang/String;I)Ljava/net/Socket;
// Two arguments, a String object and an integer, returning a Socket
// object.

void NoArgsNoResult( ) { }
// ()V
// No arguments, returning nothing

}
```

52.3.3 属性

属性是类文件结构的设计者为了允许在文件中包含关于类的附加描述信息，并且不改变其语义，而创建的一种机制。属性是动态并且结构化的修饰符，其中包括强制和可选两种影响类、类的字段和方法的特性。例如，关于一个方法的变量、参数及编译的字节码的信息包含在一个强制性的属性中，该属性称为代码(code)属性。

同样，关于使用属性来扩展.class 文件的信息，微软的 JVM 实现通过向.class 文件中增加新的属性，提供了支持与 COM 对象相互操作的能力。类加载器和 JVM 实现只需识别强制性属性即可，对其它属性可以忽略。这样，为一个 VM 所编译的类仍可以被另一个 VM 所读取(也有可能被执行)。

警告：例如，很显然，如果我们根据一个支持 COM 对象的 VM 创建了一个.class 文件，则它不可能在 Sun 的 JVM 1.0.2 下运行。

表 52.11 给出了 Sun 公司的 JVM 1.0.2 可识别的属性及简要描述。

表 52.11 Sun 公司的 JVM 1.0.2 中 Java 的.class 文件属性

属性名	强制性	级别	目的
SourceFile	否	类	包含.class 文件的 Java 源文件的名字
ConstantValue	是	字段	存放用于初始化一个本地类型字段的值
Exceptions	是	方法	定义由该方法抛出的异常
Code	是	方法	为一个方法定义的物理结构和字节码
LineNumberTable	否	代码	包含一个用于调试的对于行号表的程序计数器
LocalVariableTable	否	代码	包含用于调试的局部变量描述信息

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

当.class 文件的元素使用属性时，它们被保存在一个表中，并且其前面带有一个 16 位无符号整型数计数字段，用来存放紧跟在其后的属性个数。属性实际上是命名的可变长结构，在某种程度上类似于本节中前面所介绍的常量池中的项。每个属性都以一个固定长度的部分作为开始，后面是个数不定的一些字段。属性也可以嵌套，以便可扩充它所包含的信息。

在所有的属性定义中，前两个字段都是相同的，见表 52.12。

表 52.12　属性定义：固定部分

字段	字节数	值
Name Index	2	常量池中一个 Utf8 字符串的位置，包含该属性的常量名，与表 52.11 中所定义的一样
Length	4	一个无符号整数，包含其后面的数据的字节个数，不包含组成固定部分（Name Index 和 Length）的六个字节
Data	（Length）	与这一特定属性定义相关的实际可变长结构

注意：我们按照每个属性在.class 文件中的实际位置描述了属性的含义及结构。在这些讨论中，我们假设每个属性都以表 52.12 中描述的 Name Index 和 Length 字段作为开始。

52.4　.class 文件的结构

现在，我们已经定义了.class 文件中使用的动态元素，下面我们来看看它的实际结构。表 52.13 给出了.class 文件中字段的第一级描述。

表 52.13　.class 文件结构中第一级字段

字段	字节数	值
Magic Numder	4	这个值的作用类似于一个签名，用于保证实际.class 文件的有效性。正如此处所写的，它必须是一个 32 位的值 0xCAFEBABE
Minor Version	2	产生该.class 文件的编译器所使用的副版本号。在 JDK 1.0.2 的 javac 编译器中这一整数值当前使用的是 3
Major Version	2	产生该.class 文件的编译器所使用的主版本号。在 JDK 1.0.2 的编译器 javac 中这一整数值当前使用的是 45
Constant Pool Size	2	随后的常量池中项的个数再加一。即，这个值表示在常量池的运行版本中实际的项数，其中包括第 0 个项。这个项没有包括在表 52.14 中
Constant Pool	不定	实际的常量池项，见前面“常量池”一节
Class Flags	2	一系列位标志（下一节中定义），为类或接口定义指定的访问许可
Class Name	2	常量池中一个类引用的索引，表示本类的完整合法的名字
Superclass Name	2	常量池中一个类引用的索引，表示本类的父类的完整合法的名字。如果该值为 0，类名必须为 java.lang.Object（唯一没有直接父类的类）

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

字段	字节数	值
No. of Interface	2	本类中所实现的接口个数
Interface List	(Number * 2)	一个常量池索引数组,指向类引用项,这些项命名了本类所实现的接口。该数组的顺序必须与编译该类时所遇到的 implements 子句的顺序相同
No. of Fields	2	本类中所定义的字段(static 和 instance 型)个数
Field Table	不定	一个字段信息结构数组,见下一部分中的定义
No. of Methods	2	本类中所定义的方法(static 和 instance 型)个数
Method Table	不定	一个方法信息(Method Information)结构数组,见下一部分中的定义
No. of Attributes	2	本类中所定义的属性个数
Attribute Table	不定	包括在本.class 文件中的属性表。在这一级别上 Sun 的 JVM 1.0.2 可识别的唯一属性是前面所定义的 SourceFile 属性

可以看出,即使在最高层以上,.class 文件也是非常动态的。没有办法在顶层读取,然后再进入更深的层并读取所感兴趣的部分。在自然和物理结构上它是完全有序的,对于大多数单独的字段从其描述来看是很清晰的,唯一的例外是标志及字段和方法的嵌入数组。

52.4.1 类标志字段

类标志字段是一个 16 位的无符号整数,用于表示一组定义该.class 文件的结构和对该.class 文件的访问许可的布尔值(见表 52.14)。它们主要由 JVM 的验证通行使用,以说明是一个类还是一个接口,并标识类的可视性和可扩展性。

表 52.14 类标志值定义

位的位置 {LSb = 1}	逻辑名	应用到类	接口	设置定义
1	PUBLIC	是	是	该类可以被本包以外的其它类访问
5	FINAL	是	否	该类不可以有子类
6	SUPER	是	是	特殊情况下调用父类中的方法
10	INTERFACE	否	是	该类表示一个接口定义
11	ABSTRACT	是	是	该类或接口为抽象的,并且具有必须在子类或接口实现中编写的方法

52.4.2 字段信息结构

字段信息结构是一个第二层的信息集合,用于描述与该类的字段相关的名字、类型及访问许可(见表 52.15)。这些字段可以是 instance 型或 static 型的(类变量),也可以表示本地类型、特定的对象引用、或者这两种类型的数组。JVM 用这些信息为类定义和每个实例的数据

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在内存中分配适当的空间。

表 52.15　字段信息结构中的字段

字段	字节数	值
Field Flags	2	定义对该字段访问许可的一系列位标志
Field Name	2	常量池中一个 Utf8 字符串的索引，表示该字段的名字
Type	2	常量池中一个 Utf8 字符串的索引，以在“类型信息”一节所描述的格式表示其类型定义
No. of Attributes	2	为该字段定义的属性的个数
Attribute Table	不定	与该字段相关的属性表。在这一层上 Sun 公司的 JVM 1.0.2 可识别的唯一属性是前面定义的 ConstantValue 属性

表 52.16 中定义了与字段相关的访问标志的含义。

表 52.16　字段标志值定义

位的位置（LSb = 1）	逻辑名	应用到类	接口	设置的定义
1	PUBLIC	是	是	从该包之外的其它类可以访问该字段
2	PRIVATE	是	否	该字段只能从该类进行访问。子类或该包之外的类不能访问该字段
3	PROTECTED	是	否	只有该类及其子类可以访问该字段
4	STATIC	是	是	该字段被认为是一个类级的字段，在内存中只出现一次，由该类的所有实例共享
5	FINAL	是	是	该字段只能出现在该类的定义中，并且不能被覆盖，当初始化之后不能再给它赋值
7	VOLATILE	是	否	说明该字段的值不保证对其访问的一致性。所以对该字段编译器将不产生优化代码
8	TRANSIENT	是	否	该字段的值只有在运行过程中，当该类的一个实例处于内存时才有效。它的值在写入或读出固定存储器时被忽略

52.4.3　ConstantValue 属性

ConstantValue 是 .class 文件的字段信息结构中一个强制性属性。当在一个类中定义本地类型（非对象）字段时，使用该属性存放这些字段的初始化值（见表52.17）。

表 52.17　ConstantValue 属性独有的字段

字段	字节数	值
Value	2	常量池中一个 Integer 常量、Long 常量、Float 常量或 Double 常量的位置

Value 字段所采用的常量类型由下表决定：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

常量池类型	所保存的值用于
Integer 常量	boolean、byte、char、integer 及 short 的初始化
Long 常量	long 的初始化
Float 常量	float 的初始化
Double 常量	double 的初始化

52.4.4 方法信息结构

方法信息结构是一个第二层的信息集合，用于描述该类中一个方法的名字、签名及访问许可(见表 52.18)。方法可以是面向实例的(只能从该类的一个实例中调用)，也可以是静态属性，可以为该类的实例或接口创建内部方法表。

表 52.18 方法信息结构中的字段

字段	字节数	值
Method Flags	2	为该方法定义访问许可的一系列位标志
Method Name	2	常量池中一个 Utf8 字符串的索引，表示该方法的名字
Signature	2	常量池中一个 Utf8 字符串的索引，表示该方法的签名定义，采用“类型定义”一节中所描述的格式
No. of Attributes	2	为该方法定义的属性的个数
Attribute Table	不定	与该方法相关的属性表。在这一层上 Sun 公司的 JVM 1.0.2 可识别的属性是前面定义的 Exceptions 和 Code 属性

表 52.19 中定义了与一个方法相关的访问标志的含义。

表 52.19 字段标志值定义

位的位置 (Lsb = 1)	逻辑名	应用到类	接口	设置的定义
1	PUBLIC	是	是	从这个包之外的其它类可以访问该方法
2	PRIVATE	是	否	只能从这个类中访问该方法，子类或这个包之外的其它类不能访问该方法
3	PROTECTED	是	否	只能从这个类及其子类中访问该方法
4	STATIC	是	否	该方法被认为是一个类级的方法，不管是否存在这个类的一个实例，都可以调用该方法
5	FINAL	是	否	该方法仅出现在这个类的定义中并且不能被覆盖
6	SYNCHRONIZED	是	否	可在多线程的情况下调用该方法，并由一个监视器控制和锁定对它的访问
9	NATIVE	是	否	该方法的实现不是以 Java 字节码完成，而是使用一些其它的外部形式完成。它必须符合 JVM 的本地调用接口规范
11	ABSTRACT	是	是	该方法的签名仅在该类中定义并必须在一个子类中实现。它有效地将这个类转换为一个抽象类

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Exceptions 属性　Exceptions 是一个给定方法的.class 文件的方法信息结构中一个强制性属性(见表 52.20)。它定义了由包含这一属性的方法所抛出的一个异常列表。它们的顺序与编译该类时出现在.java 源文件中的 throws 子句的顺序相同。这一信息由类加载器和 JVM 使用,用于检查是否允许一个方法抛出一个给定的异常。

表 52.20　Exceptions 属性所独有的字段

字段	字节数	值
Count	2	Utf8 常量池项后面的表中元素个数
Table	(Count * 2)	对 Utf8 常量池项的一个索引数组

Code 属性　Code 属性是方法信息结构的一个强制性属性,它定义了源代码语句编译后的实际表示(见表 52.21)。前两个字段由 JVM 使用,以确认定义其堆栈帧所占用的内存空间数量。字节码在运行时执行,Exceptions 是在运行时进行监视和处理的,而属性(如果有的话)是在调试时使用。在 Sun 公司的 javac 编译器中,当使用 - g 选项时,将插入 LineNumberTable 和 LocalVariableTable。在 Code 属性的这些描述之后是属性的详细内容。

表 52.21　Code 属性所独有的字段

字段	字节数	值
Stack Depth	2	JVM 的表达式堆栈所允许的最大深度
No. Locals	2	定义在该方法中的局部变量(包括参数)的个数
Code Length	4	随后的字节码流所使用的字节数
bytecodes	(Code Length)	Java 字节码流,表示该方法中语句的编译版本
Exception Count	2	该方法中所捕捉的异常个数,见表 52.22
Exceptions	(Count * 8)	一个固定长度结构的有序表(表 52. 22 中描述),详细描述了该方法中每个 try - catch 子句的编码
Attribute Count	2	在随后的属性表中定义的属性的个数
Attribute Table	不定	为该方法的 Code 属性提供的属性表。目前仅支持 LineNumbeTable 和 LocalVariableTable 子属性

嵌入的异常表具有如下的格式,见表 52.22。

表 52.22　Code 属性的嵌入异常表中的字段

字段	字节数	值
PC Start	2	处理该异常的 try 块的第一个字节码
PC End	2	该异常的处理器不再活动的字节码的地址(紧跟在 try 块后的字节码)
PC Exception Handler	2	实际的异常处理器开始处的字节码位置
Exception Type	2	常量池中一个类引用常量的索引,表示实际所处理的异常

Code 属性的嵌入属性的定义在以下的部分中讨论。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

LineNumberTable 属性 LineNumberTable 是 Code 属性的一个可选属性，它包含一个对行号转换项的程序计数器表(见表 52.23)。它们是按 PC 的位置排序的，并且可以包含重复的行号引用。这一反常现象产生的原因是，代码以一般方式产生，并且在 Sun 公司的 javac 编译器创建它们时对所产生的 Java 字节码进行了优化。

表 52.23 LineNumberTable 属性所独有的字段

字段	字节数	值
Count	2	随后的行号信息表中的元素个数
Table	(Count * 4)	包含行号信息元素的一个表，见表 52.24 中的描述

实际的行号表元素具有如下的固定长度结构，见表 52.24。

表 52.24 LineNumberTable 属性的行号表中的字段

字段	字节数	值
PC Start	2	与给定行号相关的一些字节码的起始处的程序计数器位置
Line Number	(Count * 4)	产生这些字节码所来自的实际行号(相对于.java 源文件起始处)

LocalVariableTable 属性 LocalVariableTable 是 Code 属性的一个可选属性，它包含了一个描述该方法中所出现的局部变量及其相应范围的有关项的表(见表 52.25)。它们是无顺序的，并且包括了表示该方法参数的项。此处需要注意的一点是，每个非静态的方法至少包含一个参数(即使在方法的签名中没有参数)，表示该类的当前对象实例。

表 52.25 LocalVariableTable 属性所独有的字段

字段	字节数	值
Count	2	随后的局部变量信息表中的元素个数
Table	(Count * 10)	包含局部变量信息元素的一个表，见表 52.26 中的描述

实际的局部变量表元素具有如下固定长度的结构，见表 52.26。

表 52.26 LocalVariableTable 属性的局部变量表中的字段

字段	字节数	值
PC Start	2	该变量进入其作用范围的程序计数器位置
Scope Size	2	从 PC Start 开始，该变量保持在其作用范围内的字节码的个数——例如，Scope = ['PC Start' to ('PC Start' + 'Scope Size' - 1)]
Name	2	常量池中一个 Utf8 字符串的位置，其中包含该变量的常量名
Type	2	常量池中一个 Utf8 字符串的位置，其中包含该变量的类型信息(见"类型信息"一节中的定义)
Variable Slot	2	该方法的堆栈帧中保存该变量的值的插槽或偏移量

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

52.4.5　SourceFile 属性

SourceFile 属性是一个可选属性，用于最高层的 .class 文件结构，存放用来编译该 .class 文件的源文件的名字(见表 52.27)。它主要用于调试系统，可以用来搜索源文件并在必要时显示源代码行。

表 52.27　SourceFile 属性所独有的字段

字段	字节数	值
FileName	2	常量池中包含 .java 文件名常量的一个 Uft8 字符串的位置

52.5　现在可以做些什么

现在我们已经充分理解了类结构的物理格式，使用这些信息可以做很多事情，包括：

- 它可以帮助我们理解 Java 语言编译器如何使用二进制格式表示源文件信息。
- 在调试时，它可以帮助我们有效地使用 JDB。
- 当我们打算使用 Java Debugger API 实现一个自定义的调试工具时，这些信息可以帮助我们分析 .class 文件。
- 它可以帮助我们创建自己的 .class 文件阅读器。

从个人的角度来看，我们在此选择第四种情况。为了对 .class 文件阅读器需要处理的细节有一个彻底的理解，我们实现了一个 Java 应用程序来帮助完成这一任务。我们创建了一个包和工具，用来分析一个 .class 文件并将其信息转换为可显示的字符串格式。驱动程序工具称为 ClassFileDump，软件包称为 com.Que.SEUsingJava.ClassFile。

工具本身非常简单，只是用来读取一些命令行参数并将其传送给包的主类。包由 32 个类组成，包含在 8 个 Java 语言源文件中。这个包的启动类称为 ClassHeader，它有一个不带参数的简单构造器和两个基本方法。第一个基本方法称为 read，使用了一个 java.io.DataInputStream 实例参数，该实例应该与一个打开的 .class 文件相关联。read 的全部责任是装入并分析 .class 文件，它通过将输入流传送到包中的其它 31 个支持类来完成这一工作。

包中的每个类都知道 .class 文件的特定结构和属性，并且知道如何读入并转换成一个字符串。当 read 方法返回之后，该工具调用 ClassHeader 实例的 toString 方法。toString 方法利用了包中的其它两个类实例，将其各自的成员数据项转换为字符串值，然后，toString 方法将这一大型字符串返回给驱动程序工具，在这里该字符串又被发送到 System.out。

> **注意**：ClassFileDump 工具可在光盘上以两种格式找到。第一种格式为该工具和包的源文件，称为 CLASSDMP_SOURCE.ZIP。第二种格式是可执行的 Java 字节码版本，存放在一个称为 CLASSDMP_LIB.ZIP 的文件中。该文件具有加入 CLASSPATH 环境变量的适当格式。例如，如果我们将 CLASSDMP_LIB.ZIP 存放在 JDK 的 LIB 目录下，我们可以将类路径修改为如下的路径：
>
> ```
> ;c:\java\lib\classes.zip;c:\java\lib\classdmp_lib.zip
> ```
>
> 当完成这一修改后，我们可以在任何可以使用 java 命令的地方运行该工具。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

ClassFileDump 的命令行如下所示：

```
java ClassFileDump < .class filename >
```

例如：

```
java ClassFileDump ClassFileDump.class
```

该命令将导致 ClassFileDump 工具的 .class 文件的内容被发送到 System.out 控制台。我们选择将输出发送到控制台的原因是，它可以很方便地被重定向到文件中。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 53 章　Java 虚拟机的内部机制

本章内容：

53.1　JVM 的元素

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

53.1 JVM 的元素

虚拟机(VM)的概念和实现已经有相当长的一段时间了。使用这种体系结构的最早的商用环境之一是 UCSD p-System。该系统由 Kenneth Bowles 博士于七十年代在圣地亚哥南加利福尼亚大学(University of Southern California, San Diego)创建。Bowles 博士创建了一个公司 SofTek Microsystems 来销售该操作系统。事实上,p-System 是 Apple 的核心操作系统,它也是 1980 年 PC 机引入以后,IBM PC 机 PC-DOS 的替代操作系统。

与 Java 的环境一样,UCSD p-Sytem 系统是基于一种主语言(Pascal)的。UCSD p-System 包含一套原始核心库、一种与机器无关的对象文件格式、一套面向字节的伪代码、以及一个用来解释它们的 VM 定义。p-System 及其 Pascal 版本甚至有一些高级特性,如全屏幕用户界面、并发性原语以及一个称为单元的动态库机制。p-System 被引用到了许多体系中并且在纵向的软件市场上获得了广泛的成功。

那么,既然 p-System 与 Java 如此相似,为什么如今却消失了呢? 在笔者向 Sun 公司的主要技术负责人 Eric Schmidt 提出这一问题时,他只是简单地回答道"你可曾知道有哪个大学能够懂得如何经销软件吗?"这里的关键是 Java 并不是什么独特的新技术。Java 环境的成功是因为它是由一个非常成功的公司来支持的,并且还有一个更加成熟的工业来作为保证。

查看一下 Java 环境,可以看出它由五个主要部分:

- Java 语言
- 字节码定义
- Java/Sun 的核心类库
- JVM 规范
- .class 文件结构

在这些部分中,.class 文件结构、节字码定义以及 JVM 规范是实际促使 Java 技术以如此之快的速度广泛地流行(或者说是无处不在的)的原因。这样,Java 的设计者们获得了任何.class 文件到任何具有 JVM 实现的计算机/芯片上几乎是直接的移植性。不管使用什么类型的计算机/芯片来编译源文件,这种移植性都是适用的。由于 JVM 在广大泛围的硬件平台和体系结构上得到了广泛的实现,这就使得"一旦编写,随处运行"的概念变成了现实。

本部分的其它内容将介绍与 Sun 公司的 JVM 实现有关的一些技术细节。显然,有很多供应商都建立了 JVM 的实现(Natural Intelligence、Netscape、Microsoft 等)。所有这些供应商在他们的实现中都增加了一些独特的特性。但最根本最重要的一点是他们全都支持 Sun 公司的.class 文件结构、字节码定义以及虚拟机的初始规范。

53.1.1 虚拟机的体系结构

那么到底什么是虚拟机呢? 虚拟机是基于一种假想的计算机的软件概念,这种计算机具有一个逻辑指令集或伪代码,这个逻辑指令集定义了这种计算机可执行的操作。一个面向虚拟机(VM)的编译器一般情况下可以处理某些源语言,对于一个特殊的目标硬件结构,编译器所产生的并不是机器代码指令,而是基于假想计算机的指令集的伪代码流。

与此对应,另一个方面的问题是这些指令是如何得以执行的。这就是一个解释器或在 Java 领域内被称为 VM 的东西要扮演的角色。解释器实际上只是一个应用程序,这种应用程序

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友, Java群空间; 可以联系群主获取更多大型企业内部技术教程。

能够理解假想计算机所使用的伪代码语义，并将其转换为作为解释器的目标，底层硬件所使用的机器代码指令。VM 还创建一个内部运行时系统，用来支持运行这些指令时的语义实现。运行时系统还负责装入对象（或.class）文件、内存管理以及垃圾收集。

由于用于属主 VM 的硬件平台工具的不一致性，典型情况下它们是基于一个堆栈计算机的概念而实现的。堆栈计算机在指令之间传送信息时不使用任何物理寄存器，相反，它使用堆栈的帧来表示方法的状态、字节码的操作对象、方法的参数空间以及局部变量的空间等。其中有一个伪寄存器称为程序计数器——它是当前所执行指令的字节码数组的一个指针。

VM 在解释阶段的逻辑实际上是一个非常简单的循环。图 53.1 表示的是典型情况下一个基于堆栈的 VM 解释器所使用的逻辑流程序图。

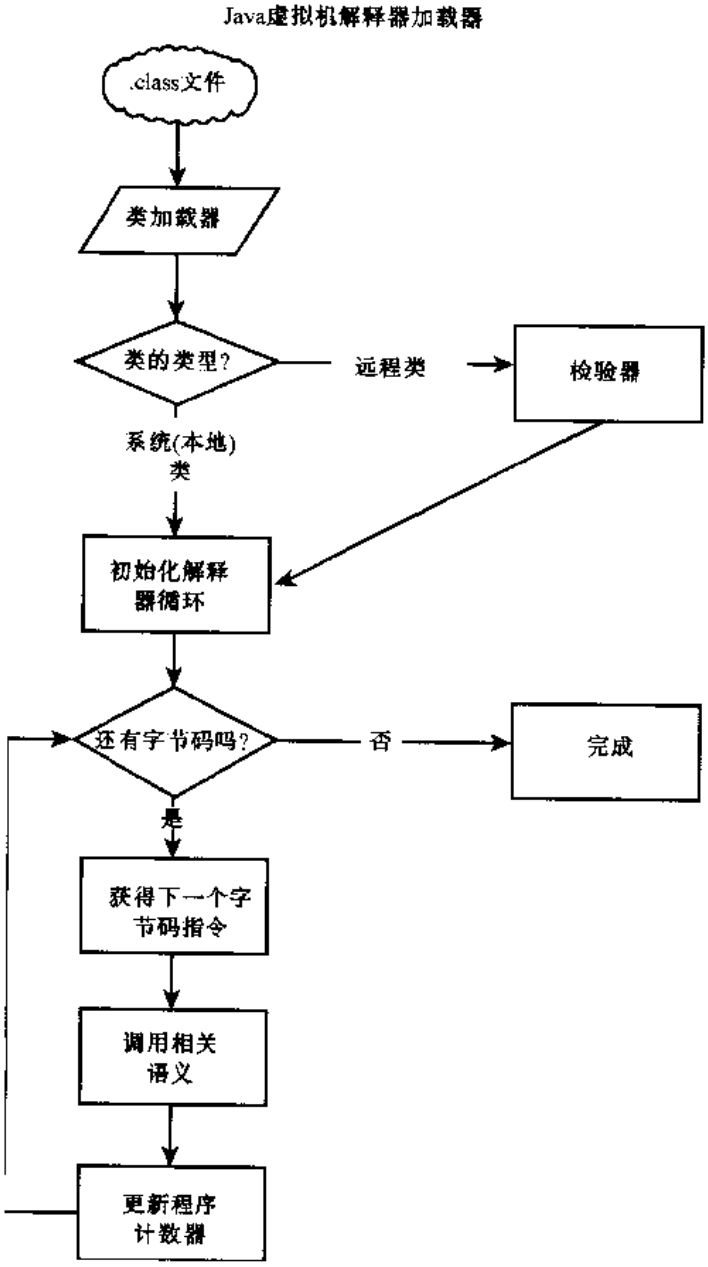


图 53.1　JVM 解释器循环

关于解释器如何实际处理字节码指令，有两点需要特别注意：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

● 大多数执行与给定字节码有关的动作的语义过程，都是从堆栈中获得其操作数，并将其结果送回到堆栈中。

● 实际上在典型的情况下字节码是带有参数的，这些参数在字节码流中紧跟在字节码自身之后，与字节码保持在一个行上。

例如，有些字节码从常量池中把值压入堆栈中，这些字节码使用常量池中值的索引作为参数。当关于这一字节码的语义完成之后，该值将处于堆栈的顶端，而程序计数器将指向该字节码紧后面的参数。下面是 Load Constant - 2 字节码的字节码流可能出现的形式：

```
ldcz index byte1 index byte2 < next bytecode(下一个字节码) >
```

其它一些并非能方便地表示出来的事情是方法调用、异常处理器以及监控器(用语言关键字 synchronized 锁定)等都是由特定的字节码来处理的。它们并不是解释器循环本身的责任。这一循环本身是很笨拙的，它只知道如何得到一个字节码并触发与之相关的语义过程。

最后，还有一些其它技术，解释器可能要使用这些技术来处理表示 VM 可执行指令的字节码流。一个通用优化的解释器技术称为线程化的解释器(不要与多线程的概念相混淆)。线程化的解释器不采用基于循环的方法遍历字节码流。相反，这种解释器会从一个字节码语义跳到另一个字节码语义，就好像使用针和线缝纫的方式一样。这种技术的最大优点是不会再有解释器的循环开销，指令执行之后只需执行一个简单的跳转即可。由于这只是一种实现方案的选择，所以它并不会影响到.class 文件的结束，它是对所有开发其自己的 VM 的人员都开放的一种选择方案。

正在变得越来越流行的另外一种优化是使用所谓即时(Just - in - Time，JIT)技术的编译器。Microsoft JVM、Microsoft Internet Explorer 4.0 以及 Netscape Navigator 4.0 都使用了 JIT 技术。JIT 编译器的思想是，并不对每个字节码流指令进行解释，而是直接将一组字节码译成针对于运行时目标系统的一组等价的机器码指令。方法的这一新的翻译过的机器码版本将会被保存起来，供以后调用这一特定方法时随时使用。这样，我们得到了基于.class 文件和字节码的可移植性。当进行完从字节码到机器码的一次性向前翻译后，也可以获得接近于本地码的执行性能。

现在我们对 VM 的体系结构有了一个更好的了解，下面让我们来看看它的内存管理方式。

53.1.2 内存管理和垃圾收集

运行系统的实现者们所面临的主要问题之一是，如何处理由内存中正在执行的程序所提出的对系统的动态内存请求。系统的设计人员必须做出选择：是让系统的用户来管理内存，还是让运行系统本身更加聪明地来自己处理这个问题。

如果我们曾经使用任何一种第三代编译语言(3GL)如 C、C++ 或 Pascal 编写过程序，那么可能经历过自己处理内存管理的“乐趣”。这些语言的运行时系统为我们提供了从一个称为堆的大内存块中分配和释放指定大小的专用内存块的源始方法。如果我们的应用程序具有相对较少的动态内存请求，那么从堆中分配应用程序所需的内存并不存在什么问题。但是大多数面向对象的应用程序都会频繁地创建和消毁相对较小的对象。

为了帮助解决这一问题，大多数的运行时系统都有一个堆管理器，动态地维护堆的状态，使尽可能多的内存可以使用。堆管理器所要解决的主要问题之一称为碎块问题，碎块是由于

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在堆中大量分配和释放不统一的小内存片的结果。一般情况下，堆的管理是通过跟踪两个内存列表来完成的：

- 自由内存块列表
- 已分配的内存块列表

当向堆管理器请求一个内存块时，管理器在自由内存块列表中搜索能够满足这一请求的一个块。大多数现代的堆管理器按照自由块的大小以递增的顺序维护自由内存块列表，这样就可使分配机制使用一种所谓的“首次适合策略”，找到第一个可用的能够满足需求的最小内存块。这一策略有助于使堆中的碎块保持在最少的程度。

为了使堆中的碎块保持在最少的程度，堆管理器使用的另外一种技术称为合并。当内存返回给堆时，一个新的内存块被放在了自由块列表中。当这一过程出现时，管理器会检查自由块列表，查看所返还的内存块是否紧挨在另一自由内存块的前面或后面。如果是，那么将这两个内存块合并，建立一个更大的内存块。

堆管理中的另一个问题是如何处理比自由块列表中所提供的每个单块都大的内存请求。这种请求需要堆管理器做一些辅助性处理，以创建更大一些的内存块。对这一问题的解决方案是一种称为压缩的技术。压缩是通过移动已分配了的内存块(两个自由块之间的内存)到堆的一端，合并所有自由块到一起的处理过程。这样就可以创建一个更大的合并自由内存块。这一过程的实际困难是运行时系统必须知道指向内存中基于堆的对象的每个变量(基于堆栈或动态的)的位置。系统必须使用它所指向的对象的新位置更新该变量。这一过程在时间开销和内存开销上都是非常昂贵的。但压缩在任何基于堆的分配系统中确实存在。

对于由用户自己负责的内存管理特别是内存释放的另一个主要问题是丢失的引用或内存垃圾的概念。这指的是那些在内存中已经分配了的对象，但指向它的引用(或指针)已经丢失，这样我们无法显式地释放它们所占用的内存。丢失引用是很容易产生的，下面的 C++ 代码给出了产生一个丢失引用的典型样例：

```
int * iArray;
//创建初始数组
iArray = new int[3];
...
//扩展该数组
if(iArrayCount == 3){
  int * tempArray = new int[6];
  for(int i = 0; i < 3; ++i)tempArray[i] = iArray[i];
  //产生一个丢失的引用
  iArray = tempArray;
}
```

一旦使用 tempArray 覆盖了 iArray，iArray 原来所指向的内存块就变成了无主块，也就是内存垃圾。在进行压缩处理时这些内存块是不能被回收的，因为它并不处于自由块列表中。解决这一问题的唯一方法是进行垃圾回收。

垃圾回收是对所有已不再需要或不再引用的已分配的内存对象，在不进行显式释放的情况下回收到自由块列表中的一种技术。它是堆管系统的一个处理过程，内存块必须以便于使用的特殊方式进行结构化，有两种常见的垃圾收集技术可以使用：引用计数和标记并清除。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

引用计数要求堆中的每个对象实例都有一个称为引用计数的字段。当将一个对象的引用分配给一个字段或变量时，该对象的引用计数被增加 1。当引用该对象的字段或变量超出其作用范围或被消毁时，该引用计数被减去 1。当一个对象的引用计数成为零时，则它不再被使用，其空间可以进行回收。这种算法的回收速度很快，但是当存在没有对象的赋值或对象作为参数传送时，将会出现性能障碍。对于所出现的每一个这种情况，引用计数在运行时都必须加以维护，这将引起运行时系统性能的普遍下降。

标记并清除算法要求每一个对象都包含一个位字段，称为标记位，或者在算法运行时要求建立一个外部数组以存放标记位。该算法从遍历堆中所有已分配的内存块开始，重新设置块的标记位。然后，检查所有指向堆中对象的字段和变量，将堆中被引用对象的标记位设置为真。最后，扫描已分配的堆中对象，找出所有未标记的对象。然后通过将未使用的对象放到自由块列表中来回收空间，或拷贝这一“活着”的对象到堆的末尾，回收原来的区域到自由块列表并进行压缩处理(标记并清除的另外一种变化形式，称为停止并拷贝)。这种算法的存储开销较低，并且不会影响运行时的执行性能，但是当内存垃圾收集器运行时可能会占用很长的时间。

现在我们看到了堆和内存管理问题的困难程度，特别是对于非 Java 的开发人员来说更是如此。下面我们看看 Java 运行时系统如何处理这些问题。

首先，JVM 使用了两个独立的堆，分别用于动态和静态的内存分配。所有的类定义、常量池以及方法表都保存在一个无垃圾收集的堆中。当一个类定义读入后，结构性的信息和方法保留在内存中。这的确增加了一些内存开销，但是对于那些在一个应用程序中进出相对频繁的类来说却提高了性能。

第二个堆被分割成了两个区域，分别向两个相反的方向增长。一个区域用于存放对象实例，另一个则包含对这些实例的“处理”。Java 应用程序中引用对象实例的字段和变量的运行时映像，实际上并不包含指向这些对象的指针，而是包含指向一个称为句柄的指针，句柄是一种特殊的具有固定长度的基于堆的内存对象。它是一个含有两个指针的结构，一个指针指向对象的方法表，另一个指针指向实际对象实例。这种设计的优点是句柄在内存中是永远不移动的，所以在压缩处理后更新指针时无需跟踪变量所指向的对象，我们只需简单地更新句柄结构的指针值即可。

堆中的对象空间是以传统方式管理的，其中有一个自由块列表和一个已分配对象列表。当实例化对象时，在自由块列表中搜索“第一个适合的块”。同样，如果可能的话，在这一阶段(与实例被放回自由块列表这一阶段相反)也可能有合并操作，以便提高垃圾收集处理的速度。另外，Java 消除了丢失引用这一问题，因为我们无需负责进行显式的对象释放。Java 语句中有一个 new 操作符但没有相应的 delete 操作符。

JVM 所使用的垃圾收集算法适用于动态堆的所有对象。当堆管理器在自由块列表中不能找到任何内存时，这个算法以同步方式运行。这一算法也可以异步地运行，这种情况下当系统有一个充足的时间段时，用于垃圾收集器的一个线程被启动。(这样做会有含糊不定的值，因为如果一个可运行的类准备好之后，将中断这种异步垃圾收集器的运行，垃圾收集线程不得不再次启动。)另外我们也可以通过调用 System.gc( )方法人为地初始化垃圾收集算法。对于频繁交互的应用程序，空闲的时间可能非常少，我们可能需要不时地手工调用垃圾收集器。

JVM 实际使用的垃圾收集器使用的是停止并拷贝(stop - and - copy)算法，但是区别并

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

不大。一般情况下当垃圾收器完成其压缩处理阶段的工作后，所有与一个对象有关的变量和字段都将需要改变。但由于所有对象引用变量都是基于句柄的，我们无需查找并更新所有指向活动对象的变量，只要更新堆中的句柄使其指向刚刚移动过的对象实例即可，这一算法的速度很快但不适用于实时应用程序。

有关 JVM 垃圾收集器的最后一个方面问题是关于 Finalizer 方法的概念。Finalizer 是一个称为 finalize 的特殊方法，它是在 java.lang.Object 基类中声明的，其原型为：

```
protected void finalize( )throws Throwable;
```

finalize 方法用于清除外部资源（如打开的文件），这种外部资源一般情况下垃圾收集过程不进行处理。垃圾收集器在垃圾收集一个对象实例之前调用 finalize 方法。问题是当我们调用 System.gc( )方法时垃圾收集并不立即执行，只是将其列入计划准备运行。内存垃圾收集器的运行优先级非常低，可能会被经常中断。事实上，垃圾收集器在应用程序结束之前是从不会收集完我们的对象的。所以，一般地说，实现 finalize 方法是否有价值是值得怀疑的。

在 finalize 方法中我们可以进行另外一种处理——“复活”一个对象实例。我们可以将 this 字段的值置于某一个其它对象的引用，阻止该对象被作为垃圾收集。在这一点上，垃圾收集器将不再调用 finalize 方法，即使当该对象实例确实已经做好了作为垃圾回收的准备时也是如此。

这就是 Java 中堆管理器的全部内容：一个堆中包含了类信息和方法的固定表，另一个堆包含了句柄表和对象实例。我们不进行任何显式的内存释放（虽然将一个未使用的对象变量设置为空将是对垃圾收集器和堆的一种暗示），并且可以通过调用 System.gc( )方法人为地运行垃圾收集器。

53.1.3　类文件验证

关于 JVM 我们所要讲的最后一个实际的算法是验证算法。验证是一个过程，当加载某些类文件时应用于这些文件。由于 Java 是面向应用程序的，而这些应用程序的程序段（.class 文件）可能分散于全球的任何一个地方，所以我们需要一种能够证明这些非本地的类可以被 JVM 正确执行的机制。缺省情况下，java 命令将对所有不是从本地硬盘加载的类进行一个验证处理。是否对一个类进行验证处理是类加载器（Class Loader）的一个功能，这是由指定在 java 命令行上的参数控制的。在支持 Java 小程序的浏览器中，所有的非系统类都要经过验证处理。

引入检验器最基本的目的是消除和阻止任何创建和传输有敌意的 .class 文件的企图。一般情况下，由于类是从网上一个不了解的源上加载的，所以要对这些类使用检验器，确认它们符合 .class 文件与 VM 规范之间的约定。检验处理的另一个好处是提高了运行时 Java 字节码的运行速度。之所以能提高执行速度是因为字节码有了很好的形式，并且在每次执行时字节码无需再检查其自己的参数。检验器也检查 .class 文件的集成情况。

验证过程可以分为四个阶段。前三个阶段在类加载时执行，第四个阶段是由实际 Java 字节码的一个子集来执行的。

第一阶段也可以称之为语法检查阶段。其主要作用是保证所加载的 .class 文件结构和语法上的完整性。在这一阶段要进行以下几部分检查：

- 魔数（magic number）。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 版本号。保证它与 VM 实现的一致性。
- 强制属性。保证这些属性存在并且有适当的格式。
- 常量池。检查它只含有有效的项目类型。

第二个阶段用于检查.class 文件"语义"上的一致性。主要检查如下几个部分：

- 访问标志。检查这些标志是否违背了类、类字段及其方法。
- 对象的继承关系。例如检查超类的字段。
- 常量池的项目。检查这些项目是否有正确的格式(例如,String 是否是一个字符串等)。

第三个阶段的工作量最为繁重,称为字节码检验器。在这一阶段要对该类中包含在每个方法定义中的实际字节码流进行数据流分析。下面的列表给出了字节码检验器的主要特性,确保了所列事项：

- 堆栈对每个所遇到的字节码处于一致状态。也就是说,它要检查表达式堆栈中是否有下溢出或上溢出的情况。
- 操作对象的参数处于适当的域中。
- 放入字段、参数及变量中的值或从它们中引用的值的类型对它们的使用是正确的。
- 传送给方法调用的参数具有正确的格式。
- 没有对未正确初始化的字段或变量的访问。

最后一个阶段实际上是在运行时执行的,包括了第三个阶段中所没有执行的检查,因为并不是所有被引用的类都需要在第三阶段内加载。对于动态地引用另一个类(字段或方法)的每一个指令,要检查它们之间的连接。然后,检查访问许可。另外,如果一切正常,对所引用的类进行实例化。再者,如果当前字节码引用了常量池中的内容,则要对该字节码进行分解处理,此时要使用字节码指令的一种特殊变化形式_ quick 替换字节码流中相应的位置。_ quick 变化形式假设所需要的值可以直接访问,无需常量池的中间求解。

53.1.4 JVM 字节码

这一部分是实际的 JVM 字节码指令的参考资料。我们在这里不可能有太多的篇幅来将实现一个 JVM 实际所需的所有细节都包括进去，但是使用这些信息我们应该能够编写一个简单的字节码反汇编程序。表 53.1 中包含了如下几列：

指令:关于相应操作码(OpCode)的文字助记符。

操作码(OpCode):实际的无符号字节码值。

Args (参数个数):字节码流中紧跟在这一指令的操作码之后的以字节为单位的操作对象个数。

指述:该指令的基本语义。

当 JVM 解释器循环运行时，实际上使用了一个逻辑寄存器——程序计数器(Program Counter),它表示了当前执行指令在字节码流中的地址。有些指令可以修改这个程序计数器,以便改变执行流程。否则,执行流程将按照字节码流的顺序逐条指令地执行下去。

表 53.1 按照操作码顺序排列的 Java 字节码指令

指令	操作码	# Args	描述
nop	0	0	空操作,不做任何动作

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

指令	操作码	# Args	描述
aconst _ null	1	0	将 null 对象引用压入栈中
iconst _ m1	2	0	将整数常量 – 1 压入栈中
iconst _ 0	3	0	将整数常量 0 压入栈中
iconst _ 1	4	0	将整数常量 1 压入栈中
iconst _ 2	5	0	将整数常量 2 压入栈中
iconst _ 3	6	0	将整数常量 3 压入栈中
iconst _ 4	7	0	将整数常量 4 压入栈中
iconst _ 5	8	0	将整数常量 5 压入栈中
lconst _ 0	9	0	将长型数常量 0 压入栈中
lconst _ 1	10	0	将长型数常量 1 压入栈中
fconst _ 0	11	0	将浮点数常量 0 压入栈中
fconst _ 1	12	0	将浮点数常量 1 压入栈中
fconst _ 2	13	0	将浮点数常量 2 压入栈中
dconst _ 0	14	0	将双精度型常量 0 压入栈中
dconst _ 1	15	0	将双精度型常量 1 压入栈中
bipush	16	1	将一个 1 字节的有符号值作为整数压入栈中
sipush	17	2	将一个 16 位的有符号值作为整数压入栈中
ldc1	18	1	将参数作为一个 8 位的常量池索引，把常量池中相关的项压入栈中
ldc2	19	2	将参数作为一个 16 位的常量池索引，把常量池中相关的项压入栈中
ldc2w	20	2	将参数作为一个 16 位的常量池索引，把所指位置的长型或双精度型值压入栈中
iload	21	1	将当前方法帧中由参数指定的索引位置上的整型局部变量的值压入栈中
lload	22	1	将当前方法帧中由参数指定的索引位置及该索引位置加 1 上的长型局部变量的值压入栈中
fload	23	1	将当前方法帧中由参数指定的索引位置上的浮点型局部变量的值压入栈中
dload	24	1	将当前方法帧中由参数指定的索引位置及该索引位置加 1 上的双精度型局部变量的值压入栈中
aload	25	1	将当前方法帧中由参数指定的索引位置上的对象引用局部变量的值压入栈中
iload _ 0	26	0	将当前方法帧中索引 0 位置上的整型局部变量的值压入栈中
iload _ 1	27	0	将当前方法帧中索引 1 位置上的整型局部变量的值压入栈中
iload _ 2	28	0	将当前方法帧中索引 2 位置上的整型局部变量的值压入栈中
iload _ 3	29	0	将当前方法帧中索引 3 位置上的整型局部变量的值压入栈中
lload _ 0	30	0	将当前方法帧中索引 0 和 1 位置上的长型局部变量的值压入栈中
lload _ 1	31	0	将当前方法帧中索引 1 和 2 位置上的长型局部变量的值压入栈中
lload _ 2	32	0	将当前方法帧中索引 2 和 3 位置上的长型局部变量的值压入栈中
lload _ 3	33	0	将当前方法帧中索引 3 和 4 位置上的长型局部变量的值压入栈中
fload _ 0	34	0	将当前方法帧中索引 0 位置上的浮点型局部变量的值压入栈中
fload _ 1	35	0	将当前方法帧中索引 1 位置上的浮点型局部变量的值压入栈中

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

指令	操作码	# Args	描述
fload_2	36	0	将当前方法帧中索引 2 位置上的浮点型局部变量的值压入栈中
fload_3	37	0	将当前方法帧中索引 3 位置上的浮点型局部变量的值压入栈中
dload_0	38	0	将当前方法帧中索引 0 和 1 位置上的双精度型局部变量的值压入栈中
dload_1	39	0	将当前方法帧中索引 1 和 2 位置上的双精度型局部变量的值压入栈中
dload_2	40	0	将当前方法帧中索引 2 和 3 位置上的双精度型局部变量的值压入栈中
dload_3	41	0	将当前方法帧中索引 3 和 4 位置上的双精度型局部变量的值压入栈中
aload_0	42	0	将当前方法帧中索引 0 位置上的对象引用局部变量的值压入栈中
aload_1	43	0	将当前方法帧中索引 1 位置上的对象引用局部变量的值压入栈中
aload_2	44	0	将当前方法帧中索引 2 位置上的对象引用局部变量的值压入栈中
aload_3	45	0	将当前方法帧中索引 3 位置上的对象引用局部变量的值压入栈中
istore	45	1	将整型值从栈中弹出，并将其存储到当前方法帧中由参数所指定的索引位置上的局部变量中
iaload	46	0	将一个数组索引和一个整型数组对象引用从栈中弹出，并将指定索引位置上的元素压入栈中
laload	47	0	将一个数组索引和一个长型数组对象引用从栈中弹出，并将指定索引位置上的元素压入栈中
faload	48	0	将一个数组索引和一个浮点型数组对象引用从栈中弹出，并将指定索引位置上的元素压入栈中
daload	49	0	将一个数组索引和一个双精度型数组对象引用从栈中弹出，并将指定索引位置上的元素压入栈中
aaload	50	0	将一个数组索引和一个数组对象引用从栈中弹出，并将指定索引位置上的元素压入栈中
baload	51	0	将一个数组索引和一个有符号的字节数组对象引用从栈中弹出，并将指定索引位置上的元素压入栈中
caload	52	0	将一个数组索引和一个字符数组对象引用从栈中弹出，并将指定索引位置上的元素压入栈中
saload	53	0	将一个数组索引和一个短整型数组对象引用从栈中弹出，并将指定索引位置上的元素压入栈中
lstore	55	1	从栈中弹出一个长型值，并将其存储在当前方法帧中由参数指定的索引 index 和 index + 1 位置上的局部变量中
fstore	56	1	从栈中弹出一个浮点型值，并将其存储在当前方法帧中由参数指定的索引 index 位置上的局部变量中
dstore	57	1	从栈中弹出一个双精度型的值，并将其存储在当前方法帧中由参数指定的索引 index 和 index + 1 位置上的局部变量中
astore	58	1	从栈中弹出一个对象引用，并将其存储在当前方法帧中由参数指定的索引 index 位置上的局部变量中
istore_0	59	0	从栈中弹出一个整型值，并将其存储在当前方法帧中索引 0 位置上的局部变量中

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

指令	操作码	# Args	描述
istore_1	60	0	从栈中弹出一个整型值，并将其存储在当前方法帧中索引 1 位置上的局部变量中
istore_2	61	0	从栈中弹出一个整型值，并将其存储在当前方法帧中索引 2 位置上的局部变量中
istore_3	62	0	从栈中弹出一个整型值，并将其存储在当前方法帧中索引 3 位置上的局部变量中
lstore_0	63	0	从栈中弹出一个长型值，并将其存储在当前方法帧中索引 0 和 1 位置上的局部变量中
lstore_1	64	0	从栈中弹出一个长型值，并将其存储在当前方法帧中索引 1 和 2 位置上的局部变量中
lstore_2	65	0	从栈中弹出一个长型值，并将其存储在当前方法帧中索引 2 和 3 位置上的局部变量中
lstore_3	66	0	从栈中弹出一个长型值，并将其存储在当前方法帧中索引 3 和 4 位置上的局部变量中
fstore_0	67	0	从栈中弹出一个浮点型值，并将其存储在当前方法帧中索引 0 位置上的局部变量中
fstore_1	68	0	从栈中弹出一个浮点型值，并将其存储在当前方法帧中索引 1 位置上的局部变量中
fstore_2	69	0	从栈中弹出一个浮点型值，并将其存储在当前方法帧中索引 2 位置上的局部变量中
fstore_3	70	0	从栈中弹出一个浮点型值，并将其存储在当前方法帧中索引 3 位置上的局部变量中
dstore_0	71	0	从栈中弹出一个双精度型值，并将其存储在当前方法帧中索引 0 和 1 位置上的局部变量中
dstore_1	72	0	从栈中弹出一个双精度型值，并将其存储在当前方法帧中索引 1 和 2 位置上的局部变量中
dstore_2	73	0	从栈中弹出一个双精度型值，并将其存储在当前方法帧中索引 2 和 3 位置上的局部变量中
dstore_3	74	0	从栈中弹出一个双精度型值，并将其存储在当前方法帧中索引 3 和 4 位置上的局部变量中
astore_0	75	0	从栈中弹出一个对象引用，并将其存储在当前方法帧中索引 0 位置上的局部变量中
astore_1	76	0	从栈中弹出一个对象引用，并将其存储在当前方法帧中索引 1 位置上的局部变量中
astore_2	77	0	从栈中弹出一个对象引用，并将其存储在当前方法帧中索引 2 位置上的局部变量中
astore_3	78	0	从栈中弹出一个对象引用，并将其存储在当前方法帧中索引 3 位置上的局部变量中
iastore	79	0	从栈中弹出一个整型值、一个数组索引以及一个整型数组对象引用，并将该整型值存放在数组中 index 指向的数组元素中
lastore	80	0	从栈中弹出一个长整型值、一个数组索引以及一个长整型数组对象引用，并将该长整型值存放在数组中由 index 指向的数组元素中
fastore	81	0	从栈中弹出一个浮点型值、一个数组索引以及一个浮点型数组对象引用，并将该浮点型值存放在数组中由 index 指向的数组元素中
dastore	82	0	从栈中弹出一个双精度型值、一个数组索引以及一个双精度型数组对象引用，并将该双精度型值存放在数组中由 index 指向的数组元素中
aastore	83	0	从栈中弹出一个对象引用、一个数组索引以及一个对象引用数组的对象引用，并将该对象引用存放在数组中由 index 指向的数组元素中
bastore	84	0	从栈中弹出一个有符号字节值、一个数组索引以及一个有符号字节数组对象引用，并将该有符号字节值存放在数组中由 index 指向的数组元素中
castore	85	0	从栈中弹出一个字符(char)型值、一个数组索引以及一个字符型数组对象引用，并将该字符型值存放在数组中由 index 指向的数组元素中

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

指令	操作码	# Args	描述
sastore	86	0	从栈中弹出一个短整型值、一个数组索引以及一个短整型数组对象引用，并将该短整型值存放在数组中由 index 指向的数组元素中
pop	87	0	从堆栈的顶端弹出一个字(word)
pop2	88	0	从堆栈的顶端弹出两个字
dup	89	0	复制堆栈顶端的字
dup_x1	90	0	复制堆栈顶端的字，并将复制的值放入向下两个字的位置上
dup_x2	91	0	复制堆栈顶端的字，并将复制的值放入向下三个字的位置上
dup2	92	0	复制堆栈顶端的两个字
dup2_x1	93	0	复制堆栈顶端的两个字，并将复制的值放入向下两个字的位置上
dup2_x2	94	0	复制堆栈顶端的两个字，并将复制的值放入向下三个字的位置上
swap	95	0	交换堆栈顶端的两个字
iadd	96	0	从栈中弹出两个整数值，把它们相加，并将结果压入栈顶
ladd	97	0	从栈中弹出两个长型值，把它们相加，并将结果压入栈顶
fadd	98	0	从栈中弹出两个浮点型值，把它们相加，并将结果压入栈顶
dadd	99	0	从栈中弹出两个双精度型值，把它们相加，并将结果压入栈顶
isub	100	0	从栈中弹出两个整型值，把它们相减，并将结果压入栈顶
lsub	101	0	从栈中弹出两个长型值，把它们相减，并将结果压入栈顶
fsub	102	0	从栈中弹出两个浮点值，把它们相减，并将结果压入栈顶
dsub	103	0	从栈中弹出两个双精度型值，把它们相减，并将结果压入栈顶
imul	104	0	从栈中弹出两个整型值，把它们相乘，并将结果压入栈顶
lmul	105	0	从栈中弹出两个长型值，把它们相乘，并将结果压入栈顶
fmul	106	0	从栈中弹出两个浮点型值，把它们相乘，并将结果压入栈顶
dmul	107	0	从栈中弹出两个双精度型值，把它们相乘，并将结果压入栈顶
idiv	108	0	从栈中弹出两个整型值，把它们相除，并将结果压入栈顶
ldiv	109	0	从栈中弹出两个长整型值，把它们相除，并将结果压入栈顶
fdiv	110	0	从栈中弹出两个浮点型值，把它们相除，并将结果压入栈顶
ddiv	111	0	从栈中弹出两个双精度型值，把它们相除，并将结果压入栈顶
irem	112	0	从栈中弹出两个整型值，把它们相除，并把余数压入栈顶
lrem	113	0	从栈中弹出两个长整型值，把它们相除，并把余数压入栈顶
frem	114	0	从栈中弹出两个浮点型值，把它们相除，并把余数压入栈顶
drem	115	0	从栈中弹出两个双精度型值，把它们相除，并把余数压入栈顶
ineg	116	0	从栈中弹出一个整型值，算术求反后将其结果压入栈顶
lneg	117	0	从栈中弹出一个长整型值，算术求反后将其结果压入栈顶
fneg	118	0	从栈中弹出一个浮点型值，算术求反后将其结果压入栈顶
dneg	119	0	从栈中弹出一个双精度型值，算术求反后将其结果压入栈顶
ishl	120	0	从栈中弹出移位数和整型值，将这个值左移若干位，移动位数由所弹出的移位数的低 5 位的值决定，并将整型结果压入栈顶

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

指令	操作码	# Args	描述
lshl	121	0	从栈中弹出移位数和长整型值，将这个值左移若干位，移动位数由所弹出的移位数的低 6 位的值决定，并将长整型结果压入栈顶
ishr	122	0	从栈中弹出移位数和整型值，将这个值算术右移(符号位扩展)若干位，移动位数由所弹出的移位数的低 5 位的值决定，并将整型结果压入栈顶
lshr	123	0	从栈中弹出移位数和长整型值，将这个值算术右移(符号位扩展)若干位，移动位数由所弹出的移动数的低 6 位的值决定，并将长整型结果压入栈顶
iushr	124	0	从栈中弹出移位数和整型值，将这个值逻辑右移(不扩展符号位)若干位，移动位数由所弹出的移动数的低 5 位的值决定，并将整型结果压入栈顶
lushr	125	0	从栈中弹出移位数和长整型数，将这个值逻辑右移(不扩展符号位)若干位，移动的位数由所弹出的移位数的低 6 位的值决定，并将长整型结果压入栈顶
iand	126	0	从栈中弹出两个整型值，执行位与(and)运算，并将结果压入栈顶
land	127	0	从栈中弹出两个长整型值，执行位与(and)运算，并将结果压入栈顶
ior	128	0	从栈中弹两个整型值，执行位或(or)运算，并将结果压入栈顶
lor	129	0	从栈中弹两个长整型值，执行位或(or)运算，并将结果压入栈顶
ixor	130	0	从栈中弹两个整型值，执行位异或(xor)运算，并将结果压入栈顶
lxor	131	0	从栈中弹两个长整型值，执行位异或(xor)运算，并将结果压入栈顶
iinc	132	0	将当前方法帧中索引(参数 1)处的整型局部变量加上一个由参数 2 指定的有符号 8 位值
i2l	133	0	从栈中弹出一个整型值，将该值转换为一个长型值，并将结果压入栈中
i2f	134	0	从栈中弹出一个整型值，将该值转换为一个浮点型值，并将结果压入栈中
i2d	135	0	从栈中弹出一个整型值，将该值转换为一个双精度型值，并将结果压入栈中
l2i	136	0	从栈中弹出一个长整型值，将该值转换为一个整型值，并将结果压入栈中
l2f	137	0	从栈中弹出一个长整型值，将该值转换为一个浮点型值，并将结果压入栈中
l2d	138	0	从栈中弹出一个长整型值，将该值转换为一个双精度型值，并将结果压入栈中
f2i	139	0	从栈中弹出一个浮点型值，将该值转换为一个整型值，并将结果压入栈中
f2l	140	0	从栈中弹出一个浮点型值，将该值转换为一个长整型值，并将结果压入栈中
f2d	141	0	从栈中弹出一个浮点型值，将该值转换为一个双精度型值，并将结果压入栈中
d2i	142	0	从栈中弹出一个双精度型值，将该值转换为一个整型值，并将结果压入栈中
d2l	143	0	从栈中弹出一个双精度型值，将该值转换为一个长整型值，并将结果压入栈中
d2f	144	0	从栈中弹出一个双精度型值，将该值转换为一个浮点型值，并将结果压入栈中
int2byte	145	0	从栈中弹出一个整型值，将该值转换为一个有符号字节型值，并将结果压入栈中
int2char	146	0	从栈中弹出一个整型值，将该值转换为一个字符型(char)值，并将结果压入栈中
int2short	147	0	从栈中弹出一个整型值，将该值转换为一个短整型值，并将结果压入栈中
lcmp	148	0	从栈中弹出一个长整型值 value2 和另一个长整型值 value1。如果 value1 大于 value2，将整型数 1 压入栈中；如果 value1 等于 value2，将整型数 0 压入栈中；如果 value1 小于 value2，将整型数 - 1 压入栈中

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

指令	操作码	# Args	描述
fcmpl	149	0	从栈中弹出浮点型值 value2 和浮点型值 value1。如果 value1 大于 value2，将整型数 1 压入栈中；如果 value1 等于 value2，将整型数 0 压入栈中；如果 value1 小于 value2，或者 value1 和 value2 中有一个值为 NaN，则将整型数 -1 压入栈中
fcmpg	150	0	从栈中弹出浮点型值 value2 和浮点型值 value1。如果 value1 大于 value2，将整型数 1 压入栈中；如果 value1 等于 value2，将整型数 0 压入栈中；如果 value1 小于 value2，或者 value1 和 value2 中有一个值为 NaN，则将整型数 1 压入栈中
dcmpl	151	0	从栈中弹出双精度型值 value2 和双精度型值 value1。如果 value1 大于 value2，将整型数 1 压入栈中；如果 value1 等于 value2，将整型数 0 压入栈中；如果 value1 小于 value2，或者 value1 和 value2 中有一个值为 NaN，则将整型数 -1 压入栈中
dcmpg	152	0	从栈中弹出双精度型值 value2 和双精度型值 value1。如果 value1 大于 value2，将整型数 1 压入栈中；如果 value1 等于 value2，将整型数 0 压入栈中；如果 value1 小于 value2，或者 value1 和 value2 中有一个值为 NaN，则将整型数 1 压入栈中
ifeq	153	2	从栈中弹出一个整型值。如果该值等于 0，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
ifne	154	2	从栈中弹出一个整型值。如果该值不等于 0，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
iflt	155	2	从栈中弹出一个整型值。如果该值小于 0，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
ifge	156	2	从栈中弹出一个整型值。如果该值大于等于 0，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
ifgt	157	2	从栈中弹出一个整型值。如果该值大于 0，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
ifle	158	2	从栈中弹出一个整型值。如果该值小于等于 0，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
if_icmpeq	159	2	从栈中弹出整型值 value2 和整型值 value1。如果 value1 等于 value2，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
if_icmpne	160	2	从栈中弹出整型值 value2 和整型值 value1。如果 value1 不等于 value2，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
if_icmplt	161	2	从栈中弹出整型值 value2 和整型值 value1。如果 value1 小于 value2，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
if_icmpge	162	2	从栈中弹出整型值 value2 和整型值 value1。如果 value1 大于等于 value2，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
if_icmpgt	163	2	从栈中弹出整型值 value2 和整型值 value1。如果 value1 大于 value2，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
if_icmple	164	2	从栈中弹出整型值 value2 和整型值 value1。如果 value1 小于等于 value2，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
if_acmpeq	165	2	从栈中弹出对象引用 value2 和对象引用 value1。如果这两个值指的是同一个对象，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

指令	操作码	# Args	描述
if_acmpne	166	2	从栈中弹出对象引用 value2 和对象引用 value1。如果这两个值指的不是同一个对象，则将两个参数的值相加，并将结果加到当前程序计数器；否则，执行下一条指令
goto	167	2	将两个参数的值相加，构造一个 16 位的值，并将结果加到当前程序计数器
jsr	168	2	将两个参数的值相加，构造一个 16 位的整型值。将该指令后面一条指令的程序计数器位置压入栈中。将所构造的 16 位值加到程序计数器，移动执行流程到子过程。在子过程的入口处，从栈中弹出返回地址，并将其保存在一个局部变量中供以后的 ret 和 ret_w 指令使用（当 JVM 处理一个 finally 块时使用该指令）
ret	169	1	使用该参数作为当前方法的帧中一个包含调用者地址的局部变量的索引。然后，将返回地址放入程序计数器中，将执行流程移回到该子过程的调用者（当 JVM 处理一个 finally 块时使用这一指令）
tableswitch	170	> 12	表示一个编译后的开关语句实现，此处所希望的情况（case）的位置保存在堆栈中。在操作码之后，可以有 0 到 3 个填充字节，以便使下一个参数具有 4 字节的边界。后面的三个参数描述了表的大小。在填充字节之后是一个 32 位的整数，表示表中 default 块的偏移量。然后是两个 32 位的值，分别表示最低和最高的允许索引值。然后是实际的表。该表是一个 32 位整型数组，包含了从该指令开始到 switch 语句中一个 case 代码块的偏移量。表中共有（最高索引值 - 最低索引值 + 1）个 32 位的表项，其第一个表项的偏移量被认为是 0。实际查找中所使用的 index 是一个整型值，必须从栈中弹出。如果索引值不在[最低索引值，最高索引值]范围内，则使用 default 块的地址。否则，从栈中弹出的 index 值中减去最低索引值，以确定表中包含新偏移量的插槽，即执行点应该转移到的位置
lookupswitch	171	> 12	表示一个编译后的开关语句的实现，这一语句是基于通过匹配一个整型键值来确定 index，该整型键值是使用表中的一个值在栈中定位的。在操作码之后，可能有 0 到 3 个填充字节，以便使下一个参数具有 4 字节的边界。后面的两个参数描述了表的大小。填充字节之后是一个 32 位的整数，表示表中 default 块的偏移量。然后是一个 32 位的值，表示构成表元素的 match/offset（匹配/偏移量）数对的个数。然后是真正的表。该表是一个 32 位整数数对的数组，包含一个与键值进行比较的值，和从这一指令开始到 switch 语句中相匹配的 case 代码块的偏移量。在实际的匹配中使用的键值是一个整数，必须从栈中弹出。如果该键值与表中的任何表项都不匹配，则使用 default 块的地址。否则，与表中的表项相匹配的索引值被加到程序计数器，执行流程从这一点开始继续执行
ireturn	172	0	从当前方法的栈中弹出一个整型值。该整型值随后被压入到调用者的方法栈帧中。控制随之返回到调用者的方法
lreturn	173	0	从当前方法的栈中弹出一个长整型值。该长整型值随后被压入到调用者的方法栈帧中。控制随之返回到调用者的方法
freturn	174	0	从当前方法的栈中弹出一个浮点型值。该浮点型值随后被压入到调用者的方法栈帧中。控制随之返回到调用者的方法
dreturn	175	0	从当前方法的栈中弹出一个双精度型值。该双精度型值随后被压入到调用者的方法栈帧中。控制随之返回到调用者的方法
areturn	176	0	从当前方法的栈中弹出一个对象引用。该对象引用值随后被压入到调用者的方法栈帧中。控制随之返回到调用者的方法
return	177	0	控制返回到调用者的方法中，没有任何结果值被压入调用者的堆栈中

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

指令	操作码	# Args	描述
getstatic	178	2	从类的静态字段中获取一个值。将两个参数的值相加，创建一个 16 的偏移量，指向常量池中一个字段引用表项。解析该类和字段，确定该值的宽度和它在类中的偏移量。根据所知的该值的宽度，从类的静态字段区域中提取该值，并将其压入栈顶
putstatic	179	2	将一个值置于一个类的静态字段中。将两个参数的值相加，创建一个 16 位的指向常量池中一个字段引用表项的偏移量。解析类和字段，确定该值的尺寸及其在类中的偏移量。根据所知的该值的尺寸，从栈中弹出该值。然后，将该值置于该类的静态字段区域中由该字段的信息所确定的偏移量处
putfield	181	2	将一个值置于一个对象的非静态字段中。将两个参数的值相加，创建一个 16 位的指向常量池中一个字段引用表项的偏移量。解析该类和字段，确定该值的尺寸及其在对象中的偏移量。根据所知的该值的尺寸，首先从栈中弹出该值，然后弹出实际的对象引用。随后将该值置于该对象引用中由该字段的信息所确定的偏移量处
getfield	182	2	从一个对象的非静态字段中获取一个值。将两个参数的值相加，创建一个 16 位的指向常量池中一个字段引用表项的偏移量。解析该类和字段，确定该值的尺寸及其在对象中的偏移量。然后，从栈中弹出实际的对象引用，从对象引用中由字段信息所确定的偏移量处提取该值，并将其压入栈顶
invoke-virtual	182	2	基于动态类型查找，激活堆栈中对象引用的一个实例方法。将两个参数的值相加，创建一个 16 位的指向常量池中一个方法引用的表项的偏移量。动态地解析该类、方法签名以及方法的字节码位置，确定参数的个数以及需要从栈中弹出的尺寸。然后，从栈中弹出有关参数，再弹出包含被调用方法的类的对象引用。对象引用和参数(按顺序)变成了为被调用方法所创建的新帧中的首批局部变量。最后，控制传送到该方法
invoke-nonvirtual	183	2	基于编译时类型查找，激活堆栈中对象引用的一个实例方法。其处理逻辑与 invokevirtual 相同，只是该类的信息已经被解析
invokestatic	184	2	激活一个类的静态方法。其处理逻辑与 invokevirtual 类似，只是此处堆栈中参数的后面不再有对象引用(因为静态方法不需要一个实例化的这种类型的对象)
invoke-interface	185	4	激活一个对象的接口方法。处理逻辑与 invokevirtual 类似，只是此处方法的参数个数是由操作码的第三个参数提供的。第四个参数作为保留，尚未使用
new	187	2	根据由参数定义的类类型创建一个新的对象。将参数的值相加，创建一个 16 位的指向常量池中一个类引用的表项。解析该类信息，为该类创建一个新的对象引用。然后将该对象引用压入栈的顶端
newarray	188	1	分配一个新的数组，其中包含来自 Java 本地类型之一的元素。待分配的元素个数在这一操作码的入口处的堆栈中。这一操作码的参数可以是下列类型指定符之一：boolean，4；char，5；float，6；double，7；byte，8；short，9；int，10；long，11
anewarray	189	2	分配一个新的数组，其中包含对象引用元素。待分配的元素个数在这一操作码的入口处的堆栈中。这一操作码的参数加在一起时，创建了一个 16 位的常量池索引，指向将要被数组元素引用的类类型
athrow	191	0	抛出一个异常。堆栈的顶端必须包含一个对象引用，该对象引用是 Throwable 的一个子类。所指定的异常对象从栈中弹出并抛出。抛出一个异常的处理过程需要在当前方法的帧中搜索一个适当的异常处理器。如果找到一个这样的异常处理器，则程序计数器被设为处理器的第一个字节码地址。否则，弹出该方法帧，异常被再次抛出给这个方法的调用者

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

指令	操作码	# Args	描述
checkcast	192	2	对给定的位于栈顶的对象引用类型，检查一个强制类型转换操作的有效性。将参数的值相加，创建一个 16 位的指向常量池中一个类引用表项的索引。解析该类的信息，将栈顶的对象引用类型与由常量池表项指定的类类型比较。如果栈顶端的对象是在常量池中找到的类或其一个子类的一个实例，继续执行下一条指令。否则，抛出一个 ClassCastExcepotion 异常
instanceof	193	2	根据参数验证一个对象是否具有指定的类型。将参数的值相加，创建一个 16 位的常量池索引，指向一个类引用表项。解析该类的信息，并从栈中弹出该对象引用。将该对象的类型与由常量池表项所指定的类类型比较。如果栈顶的对象是常量池中找到的类或其一个子类的一个实例，则将整型值 1 压入栈中；否则，将值 0 压入栈中
monitorenter	194	0	进入当前字节码流的一个监视部分，并从栈顶弹出该对象引用。尝试对该对象引用分配一个独占锁定。如果另一个监视程序已经锁定了该对象，等待其锁定被释放。如果锁定了该对象，则继续。否则，对该对象分配一个新的独占锁定
monitorexit	195	0	退出当前字节码流的一个监视部分，从栈顶弹出该对象引用，并删除对该对象引用的独占锁定。如果没有其它线程锁定该对象，那么将通知所有其它正在等待该对象的线程，告知它们该对象现在可以使用了
wide	196	1	为局部变量的加载、存储及增量操作码提供一个 16 位索引。通过将(arg1)中的 8 位值与字节码流中该操作码之后的参数中的索引值相加而实现
multia－newarray	197	3	分配一个新的多维数组，其中包含对象引用元素，每一维的待分配元素个数在该操作码入口处的堆栈中。这一操作码的前两个参数相加时，创建一个 16 位的常量池索引，指向数组元素将要引用的类类型。第三个参数是该数组将包含的维数
ifnull	198	2	从栈中弹出一个对象引用。如果该对象引用为 null，则将两个参数值相加，并将结果加到当前程序计数器中；否则，执行下一条指令
ifnonnull	199	2	从栈中弹出一个对象引用。如果该对象引用不是 null，则将两个参数值相加，并将结果加到当前程序计数器中；否则，执行下一条指令
goto _ 2	200	4	将四个参数的值相加，构造一个 32 位的值，并将结果加到当前程序计数器中
jsr _ w	201	4	将四个参数的值相加，构造一个 32 位的整型值。将紧随这一指令之后的指令的程序计数器位置压入栈中。在进入子过程的入口处，从栈中弹出返回地址，并将其保存在一个局部变量中，供以后在 ret 和 ret _ w 指令中使用。将该 32 位的值加到程序计数器中，使执行流程移到子过程(当 JVM 处理 finally 块时使用这一指令)
breakpoint	202	0	停止执行并将控制交给 JVM 的断点处理器
ret _ w	209	2	将两个参数的值相加，创建一个 16 位的索引，指向该方法帧中包含调用者返回地址的一个局部变量。然后将返回地址置于程序计数器中，移动执行流程到这一子过程的调用者(当 JVM 处理 finally 块时使用这一指令)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第九部分　JavaScript

第 54 章　Java 与 JavaScript
第 55 章　JavaScript 入门

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 54 章　Java 与 JavaScript

本章内容：

54.1　Java 与 JavaScript
54.2　JavaScript 并非 Java
54.3　解释与编译
54.4　基于对象和面向对象
54.5　强类型与松散类型
54.6　动态绑定与静态绑定
54.7　磁盘访问限制
54.8　不同功能(范围限制)及与 HTML 的代码集成
54.9　快速发展与相对稳定
54.10　库
54.11　JavaScript 和 Java 集成

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

54.1 Java 与 JavaScript

如果我们从头至尾学习了本书的内容，那么现在应该对 Java 有了很全面的了解，知道了 Java 到底能为我们做些什么。本书正在接近于尾声，如果我们打算建立 Web 上的 Java 小程序，那么现在是介绍另外一种我们会遇到的语言的时候了。下面的几章将会使我们对 Netscape 新的描述语言 JavaScript 的特性有一个更好的理解。我们将学习 JavaScript 究竟与 Sun 的 Java 有什么区别，我们还要学习如何使用 JavaScript 奇迹般地增强 Web 页的性能。我们将会看到，JavaScript 和 Java 是两种完全不同的语言，但它们可以在 Web 浏览器环境下一起工作，创建出具有高度交互功能的页面。

程序员会常常被 Java 和 JavaScript 相似的名字搞糊涂。如果我们给一个朋友说“我正在使用 JavaScript 编程”，那么这位朋友很可能会有这样的回应“噢，也许你能帮助我解决一下这一 Java 小程序的...”。

Java 和 JavaScript 常常被误认为是来自同一种语言的不同部分。实际上根本不是这样。虽然它们有着类似的名字，但它们之间却存在着相当大的差别。首先一点是它们的起源。1991 年 6 月前后，Sun Microsystem 公司开发出 Java，并且最初称之为 Oak，1995 年 5 月在 SunWorld'95 上正式发布(在这之前进行了大量的改进，并且改变了名称)。而 JavaScript 是由 Netscape Commumications 公司开发，最初称为 LiveScript。Sun 公司更改 Oak 为 Java 是由于版权方面的原因，因为已经有另外一种语言叫做 Oak 了，Netscape 更改其 LiveScript 为 JavaScript，是经过了与 Sun 公司签订协议，决定把 JavaScript 开发成为一种面向非程序员的语言。JavaScript 的首次发布是随 Netscape 2.0 面世的。

在深入研究这两种语言的某些区别之前，让我们先来概括介绍一下两者之间的主要差别，然后再逐一地深入讨论。表 54.1 中列出了 Java 与 JavaScript 之间的一些主要区别。

表 54.1 JavaScript 和 Java 的比较

JavaScript	Java
由 Netscape 公司开发	由 Sun 公司开发
代码由客户(Web 浏览器)解释	代码被编译，并且在客户端运行之前存放在服务器上
基于对象。可以建立对象，但它不是类，不能继承	面向对象。所有的一切都是一个可以使用继承的类的扩展
数据类型不需要声明(松散类型)	数据类型必须声明(强类型)
运行时检查对象引用(静态绑定)	编译时检查对象引用(动态绑定)
限制磁盘访问(在写文件之前必须请求)	限制磁盘访问(访问级别由用户设置，不能自动写到磁盘上)
脚本限于 Web 浏览器功能	编译后的代码既可以作为一个 Web 小程序运行，也可以作为独立的应用程序运行
脚本使用 HTML 元素(标签)工作	可以处理多种元素(如音频、视频等)
语言的功能仍在迅速地发展变化	大部分的主要变化已经完成
没有可以用于建立 Web 的标准代码库	Java 有大量与语言捆绑在一起的库

54.2 JavaScript 并非 Java

我们应该了解的第一件事情就是 JavaScript 并不是 Java。JavaScript 程序员会经常碰到

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java 问题，即使某个论坛（如新闻组）明显是关于 JavaScript 的，这几乎变成了一种习惯。

我们可以编写出一个 JavaScript 脚本，而不使用任何一个 Java 小程序，JavaScript 采用 Java 作为其名字的原因（除了与 Sun Microsystem 公司的协议之外）是这种语言与 Java 有着类似的语法。Netscape 也认识到了采用 Java 之后的潜能，通过使用该名字来增强 JavaScript 的影响。如果我们使用 Java 编写过程序，则会发现使用 JavaScript 非常直观和容易，对我们来说没有多少新东西需要学习。然而 JavaScript 的最妙之处在于我们不需要有任何使用 JavaScript 的经验就可以迅速地创建出有用的脚本。

下面的例子给出了一个在 Java 和 JavaScript 中使用相似名称的例子，让我们来看一下两种语言是如何处理从一个指定的对象调用一个指定函数这一问题的。

在 Java 中假设我们有一个称为 MyClass 的类，它包含一个称为 MyMethod 的方法。我们可以使用下面的方式来调用 MyMethod 方法：

```
foo = new MyClass( );
result = foo.MyMethod(parameter1,parameter2);
```

在 JavaScript 中，我们也可以做同样的事情。如果我们有一个以如下方式定义的函数称为 MyObject：

```
function MyObject(parameter){
this.firstone = parameter;
this.MyFunction = MyFunction;
}
```

假设 MyFunction( )已经在前面定义了，那么可以以如下方式调用 MyFunction：

```
foo = new MyObject(parameter);
foo.MyFunction(someparameter);
```

在第一部分中，我们创建了两个属性（对象中信息的基本插槽）称为 firstone 和 MyFunction。在第二部分中，我们看到了如何创建一个对象的特殊实例，以及如何使用新对象的方法。

这两种语言之间有许多像这个例子一样的类似之处。关于 JavaScript 的语法，详见下一章。

参见第 55 章“JavaScript 入门”。

54.3　解释与编译

JavaScript 代码几乎总是放在将要执行它的 HTML 文档中。当我们加载一个包含 JavaScript 代码的页时，Web 浏览器所包含的内置解释器将接受所加载的代码并执行所有指令（有时候当我们在窗口中看到任何信息之前会立即执行）。这就是说我们通常可以使用“View Source（观察源代码）”（在 Netscape 的菜单中选择 View 及 Document Source）查看 HTML 文档中的代码。

JavaScript 使用 <script> . . . </script> 标签，这类似于 Java 的 <applet> . . . </applet> 标签。<script> . . . </script> 之间的任何内容都将被 Netscape 的 HTML 分析器所忽略，但要传送到 JavaScript 解释器。对于不支持 <script> . . . </script> 标签的 Web 浏览器来说，将 JavaScript 代码包含在注释中是一种习惯。程序清单 54.1 中给出一个例子。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<html>
<head>
<title>JavaScript Hello! </title>
<script language="JavaScript">
<! -- // to hide from old browsers
var textData = "Hello World!";
function showWorld(textInput) {
document.write(textInput);
}
//to ignore end comment -- >
</script>
</head>
<body bgcolor=white>
<script language="JavaScript">
<! --
showWorld(textData);
// -- >
</script>
</body>
</html>
```

程序清单 54.1 的脚本说明了如何在 HTML 的 < head > 和 < body > 元素中使用 JavaScript。该脚本首先在 < head > 元素中加载了一个函数(所有代码都是由上而下阅读——当涉及到其它的代码段时一定要记住这一点,所以我们不必关心那些尚未加载的代码。)当浏览器遇到 < body > 元素中 showWorld(textData)行时,将显示出 textData 的文本值——此处为"Hello World!"。

如果要使用 Java 来做类似的事情,我们可以在一个编辑器中编写 Java 代码,将其编译为.class 文件,并将该文件放到服务器上。然后使用熟悉的 < applet > . . . < /applet > 标签,将这一小程序嵌入到 HTML 文档中。例如,Java 代码如下所示:

```
public class HelloWorld extends java.applet.Applet{
public static void main(String args[ ]){
System.out.println("Hello World");
}
}
```

HTML 代码如下所示:

```
<html>
<head>
<title>Java Example</title>
</head>
<body>
<applet code = "HelloWorld.class" width = 150 height = 25>
```

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
</applet>
</body>
</html>
```

不使用编译器编译代码而使用浏览器解释代码，然后再使用浏览器运行代码，这种方法的好处主要是，我们——作为一个 JavaScript 程序员——可以通过浏览器很快地修改代码并检验其结果。如果使用 Java，则必须经过修改代码、编译代码、再次加载代码（如果在自己的 Web 服务器上测试的话），最后才能在浏览器中看到结果。一般来说，解释代码的执行速度不会像编译代码那样快，但是在 JavaScript 有限的范围之内，我们可能会看到脚本比相应的 Java 程序（具有同样功能，如滚动的文本标志）运行稍快一些的情况。

在 HTML 文档中使用 JavaScript 代码的缺点是，我们所编写的任何代码对于其它可以访问这一页面的用户都是公开的——甚至包括那些想要把我们的代码使用到他们的项目中的用户。对较小的脚本，这并不是什么大问题，对于一个较大的项目也不会成为问题——如果我们不介意自己的努力被使用到其它的页中或被其它人修改了的话。如果我们的项目很大并且需要保持代码为私有代码，可以考虑使用 Java 而不是 JavaScript。最近随着 SRC 属性的实现，我们可以将脚本从 HTML 页中抽出来，将其保存在自己的文件中。这样就大大增加了那些需要重复使用的脚本的用途。总而言之，JavaScript 程序员可以很快地在浏览器中看到他们的修改结果，这要比 Java 中编写代码/编译/加载/观察这样一个过程方便得多。

JavaScript 这种解释性的本质特性之一是，我们可以立即测试一条语句的结果（如 eval(7 * 45/6)或 document.write("hi!")）。如果我们正在使用 Netscape 的 Navigator 2.0 或其更高的版本，可试着在 URL 窗口中输入 javascript: 或 mocha: 。我们将会看到浏览器窗口变成了解释器模式，其中一个较小的框架上面有一个较大的框架。我们可以在较小的框架中输入 JavaScript 命令，在较大的框架中观察显示结果（见图 54.1）。

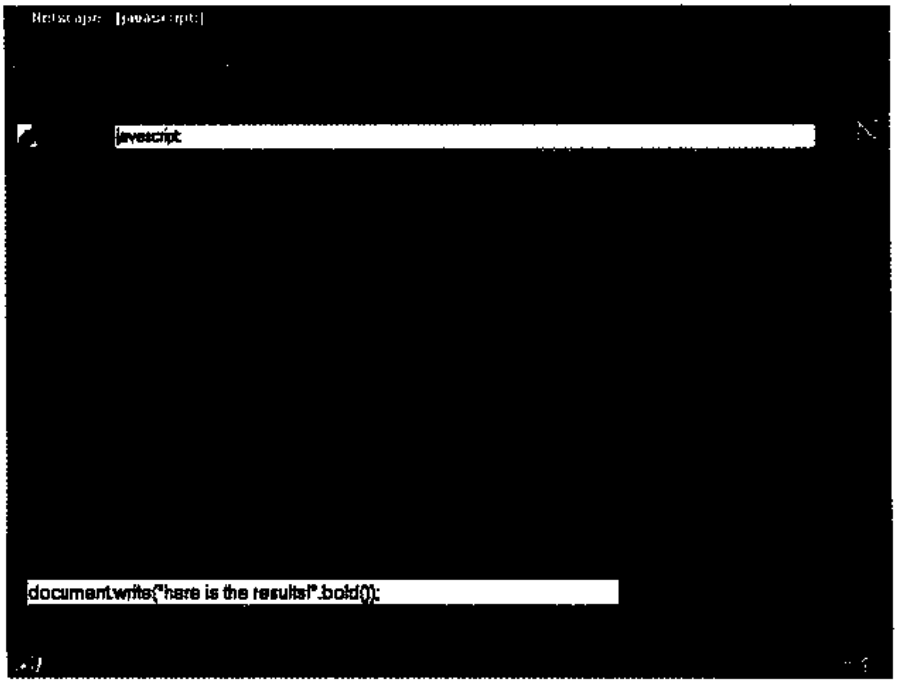


图 54.1　JavaScript 解释器运行在底部窗口中输入的语句，并在上部的窗口中显示其结果（Mac 和 Windows 版本的显示有所不同）

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

54.4 基于对象和面向对象

JavaScript 充分地采用了 Java 的整体语言结构，但是它缺乏 Java 作为一种面向对象的语言的许多特性。JavaScript 具有内置的对象(如 Navigator 、Window 或 Date)，这些对象可以访问浏览器的许多元素，如窗口、链接及图像等。一般情况下，Java 不能访问浏览器的任何一种元素，并且限制在包含它的区域(或绑定框)，以及随后创建的 Java 窗口中。

JavaScript 允许我们创建新的对象，这种对象实际上是函数。JavaScript 中的对象实际上并不是真正的对象，因为它们并不实现继承等 Java 中对象所具有的特性。例如，我们不能创建一个继承 JavaScript 的对象 Window(JavaScript 的顶层对象)的特性的新类 MyWindow。

这种限制初看起来似乎有些过于严厉，但我们仍可以在 JavaScript 中创建许多有用的函数，执行许多 Java 小程序可以执行的功能。

正是由于这些内置的对象，才使 JavaScript 在访问和操作基于浏览器的属性时，表现得非常出色。例如访问和操作当前时间、给定元素的值、浏览器中的第三个窗口、以及五次单击之前所访问过的链接等。

使用 Java，程序员可以编写出完成计算机上所允许的任何操作的独立应用程序或 Java 小程序。但是 JavaScript 填充了一个主要空隙(至少在 Java 小程序与 Web 浏览器之间)，它通过作为 Java 与浏览器之间的一种粘和剂，实现了二者之间的通信。使用 JavaScript，我们可以在 HTML 文档中向一个表单的字段中输入信息，在这个页面上的 Java 小程序可以使用所输入的内容显示新的信息。JavaScript 允许 Java 小程序访问 HTML 页面的属性，也允许非程序员访问 Java 小程序的各个部分——例如公共变量等。

虽然 JavaScript 不支持继承，但它却有一个很有意思的新特性称为原型。使用原型我们可以向(使用 new 语句)所创建的任何对象添加新属性，甚至可以向已有的内置对象添加新属性。这就是说即使在对象已经定义之后，我们仍可以扩展已有对象的实例。

例如，我们有一个对象 House，它具有 Light、GarageDoor 及 BurglarAlarm 属性。我们可能已经有了它的一个实例称为 myHouse。但是现在我们需要向 House 对象增加 ChimneySmoke 属性，我们无需重新定义所有的对象，只要使用下面的语句即可：

```
House.prototype.ChimneySmoke = ChimneySmoke.
```

现在，称为 myHouse 的实例就可以使用下述方式访问新的属性 ChimneySmoke 了：

```
myHouse.ChimneySmoke
```

54.5 强类型和松散类型

在 JavaScript 代码中声明变量时无需指定变量的数据类型。这种松散的数据类型意味着 JavaScript 程序员在创建和操作变量时更加方便了，他们不需要再担心数据到底是 int、float 还是其它类型。

在 Java 中，我们必须在使用变量之前，显式地声明变量的数据类型，这种情况称为强类型。强类型有助于 Java 代码的全局稳定性；但却给新程序员带来了不少麻烦。JavaScript 可以忽略这一点，它假设第一次所指定的值的类型就是打算使用的类型。例如，如果我们有一个

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

变量 HouseType，并且赋给了一个“Victorian”值，JavaScript 假设 HouseType 的类型将一直是字符串类型，并不会对于没有特别设置其类型是一个字符串类型而报错。然而如果我们第一次为 HouseType 赋值为 4，那么 JavaScript 将认为它的类型为 INT。总之，这种松散数据类型使得脚本的编写变得庚快更简单了，取消了对变量声明的不必要检查。然而，我们必须谨慎的为每一个变量赋以正确类型的值。

这种松散类型正是说明 JavaScript 追求简化编程过程的一个方面。由于它是直接面向非程序员或初级程序员的，其开发人员的目标是，在尽可能简化语言的同时仍保持尽可能大的灵活性。其它松散类型语言的例子还有 HyperTalk、dBASE 以及 AppleScript。

54.6　动态绑定和静态绑定

由于 JavaScript 是在客户端浏览器上解释执行的，所以其对象引用检查是动态的，这一点与 Java 在编译时的静态绑定正好相反。绑定的意思是将一个变量名与一个类型捆绑在一起——不管是通过显式地声明一个变量的数据类型静态地绑定，还是通过由计算机在编译或运行时所确定的隐式关系动态绑定。正是由于 JavaScript 的动态绑定本质，JavaScript 的对象是在运行中建立，甚至可以根据某些外部因素（时间、客户响应等）改变脚本的功能。Date 对象常用来获取 Web 浏览器的当前日期、时间及星期等信息，但这个对象是在解释 JavaScript 代码时才进行创建，以便能获取正确的信息。

Java 程序采用静态绑定，一般来说具有较好的稳定性，因为编译代码的整个过程已经由 Java 编译器完成了，所有错误或疏忽的对象引用已经得到了纠正，当我们需要在用户的机器上快速加载和运行一个给定的应用程序时，这是一个优点。

54.7　磁盘访问限制

由于许多方面的原因，安全问题成了当今 Internet 和 intranet Web 领域内的热门话题。人们在使用 Web（或 Internet 应用程序如电子邮件）时最大的担心之一是怀有恶意的程序进入他们的计算机，毁坏或危害他们的敏感性数据。在这方面 Java 具有比较全面的处理安全性问题的方法，它允许其它程序在自己的计算机上做一些有用的操作，同时使这些应用程序的操作与自己的敏感性文档保持隔离。Java 小程序一般情况下不允许进行写硬盘操作，如果存在这种情况，也只能是以限制非常严格的方式进行。

由于 JavaScript 可以控制浏览器的许多方面，所以一个时期以来人们认为 JavaScript 不如 Java 安全。这主要是由于 JavaScript 早期版本的程序陷阱所致，它允许将包含我们的书签及电子邮件地址的文件内容通过一种隐藏表单发送到另一个远程站点，或者获取我们机器上的全部文件列表。现在 JavaScript 已经解决了这些问题，并且就大部分情况而言，它与在浏览器上运行的 Java 小程序具有同样的安全性。

注意，JavaScript 能够进行写硬盘操作——这是 Web 浏览器的一个基本特性，以便进行写缓存或保存从 Web 上所下载的文件。然而，下载一个文件时要求我们特别地单击一下对话框中的接收（Accept）按钮。在这种情况下，由于 JavaScript 允许使用浏览器创建文件以保存信息供以后使用，所以 JavaScript 比 Java 更加通用。

如果谈到有敌意的代码会毁坏我们的机器，应该知道，长期以来到处都存在被病毒感染的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

可能性。Java声称它很安全,这也只是Java更多地关注了一些安全性问题而已。不久的将来,随着JavaScript的tainting(一种标识数据的系统,以便使被标识的数据不能通过JavaScript使用表单或邮件发送到其链接对象上)和Java的数字签名(代码或信息的源或身份的电子验证)的实现,我们将可以检验代码——Java或JavaScript——是否来自于一个可信任的源。这将是一件很好的事情,因为我们将可以允许Java执行一些更敏感的任务,如更新Oracle数据库内容,发送或自动安装我们机器上的软件的更新版本等。

还应该注意,Web浏览器(以及JavaScript通过浏览器)也允许把信息写入一个称为cookie的文件中。这个文件存放在我们的机器上,允许Web浏览器存放一些有关我们自己的信息,以备我们再次访问该站点时使用。只有向cookie文件写入信息的站点可以检索该文件。这种文件通常只用于维护一些不同页面之间需要传送的临时性信息(例如用户ID或我们是来自于弗罗里达州等这类事实)。

54.8 不同功能(范围限制)及与HTML的代码集成

JavaScript仅限于Web浏览器中(不管是Netscape的Navigator,还是微软的Internet Explorer)。而Java则既可以作为一个独立的应用程序(如Microsoft Word)运行,也可以在浏览器中作为一个小程序运行。正是Java这种多样性,使得它很快地适应了Web的需求。Java最初的目的是作为操作系统运行,并控制顶层框的设置或其它小通信设备。现在,作为一种可选的编程语言,Java可能最终超越C++(如果不是对所有程序员来说的话,由于其速度的提高,至少对Inernet程序员是这样),它具有更强的功能和多样性,可以在许多不同的操作系统下运行。

JavaScript是一种较小的语言,仅限于Web浏览器程序员使用。JavaScript使得Web程序员可以访问和修改所有的HTML标签、表单元素、窗口元素、图像、书签、链接以及Web浏览器中的锚点。它也支持程序员创建可以根据诸多因素如时间或用户概貌等信息进行响应或修改的Web站点(见图54.2)。JavaScript允许Java小程序、脚本以及浏览器插件之间进行相

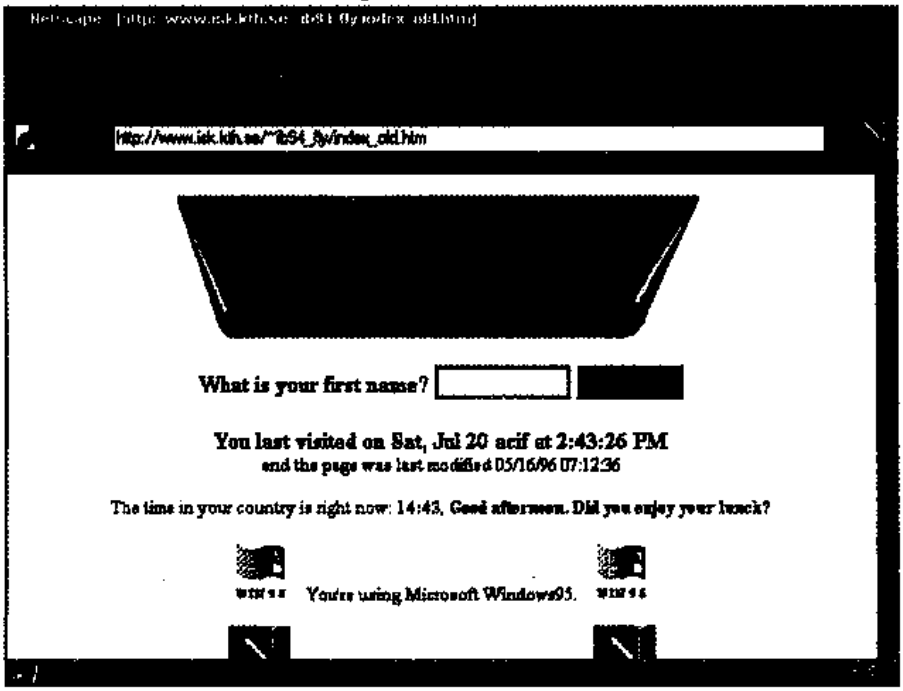


图54.2 显示基于时间信息的页面

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

互通信。这实际上是来自于 Netscape 公司的一套技术称为 LiveConnect，它需要在 Java 小程序或插件中增加一些附加代码，以便能“认识” JavaScript。

JavaScript 可以动态地创建 HTML 文件，即时修改一个页面的属性(例如背景颜色)，它还允许客户端的机器执行许多传统情况下只能通过 CGI 运行的功能，例如 TicTacToe 游戏。JavaScript 允许表单输入有效性验证，在它们被发送回浏览器之前，检查出错误的响应，而在 Java 中要做到这一点要困难得多。实际上，为了使用 Java 实现 JavaScript 的这种功能，我们必须在自己的代码中重新建立一个小型的浏览器——这是一种效率很低的解决办法。

另外，JavaScript 可以直接地集成到 HTML 文档中。事件处理器例如 onClick 可以修改浏览器的行为，而不只是可以访问新的文档。我们可以利用浏览器已经提供的 GUI 以及 HTML 表单或 TABLE 元素所提供的已有的布局功能，使用 JavaScript 建立简单的计算器。也就是说，我们使用浏览器除了可以访问文档之外，还可以做许多事情。我们可以把浏览器作为任何一种应用程序的前端。如果使用 Java，虽然它有 GUI 功能，但必须手工激活并实现这些功能，这也许需要自己编写大量的程序。然而在 JavaScript 中，浏览器处理了大部分 GUI 问题，所以我们可以集中精力来完成创建方面的工作。

下面是集成 JavaScript 语句到 HTML 的一些例子。在程序清单 54.1 中我们看到了如何通过 < script > 标签创建脚本的例子，我们可以直接在自己的 HTML 中嵌入 JavaScript 语句：

```
< a href = "" onMouseover = "alert('you noticed me!');" > pass
your mouse over here for a message! </a>
```

此处，当鼠标经过这一超链接文本时，不是在状态条上显示一个 URL，而是弹出一个新的对话框，其中显示出“you noticed me!”文本。在 Java 中要做到这一点，我们必须创建一个小程序在屏幕上画出链接文本，并且需要编写监视鼠标位置的代码(也许要作为一个独立的线程)，还必须编写更多用来创建和消毁结果对话框的代码。显而易见，对于那些临时性 Web 设计者来说，JavaScript 方案是一种更容易实现的方案。另外，使用 LiveConnect 可以使 Java 小程序通知 JavaScript 打开窗口，或完成其它一些工作，无需再编写附加代码。

请看如下的 JavaScript 例子：

```
< img width = "&{imgwidth}";%"height = 40 >
```

此处，JavaScript 允许我们使用 JavaScript 表达式为 HTML 属性设置一个值。在这一例子中，如果我们已经把 imgwidth 定义为 50，则所显示的图像的缩放宽度将是窗口尺寸的 50%。另外，如果我们在 Java 中完成这一任务，将要编写相当多的代码——只是用来模仿相同的功能。即便如此，我们也不能方便地与其它小程序共享 imgwidth 的值。

总之，直接把 JavaScript 代码集成到 HTML 源中的能力，可使 Web 程序员快速利用浏览器的内置式 GUI。临时性的脚本编写人员既可以增加他们的 HTML 经验，也可以对 Java 有所了解。Java 允许我们创建可以运行于许多操作系统的有趣的应用程序，只要这种系统上装入了 Java 解释器。

Java 可以创建与 C++ 同样广泛的应用程序，这种能力无需再争辩。然而，如果我们作为一个 Web 页面的设计者，没有太多的时间去学习 Java 编程，则可能会发现 JavaScript 是一个很有用的语言。另一方面，如果我们不喜欢在 HTML 中使用只能在两种浏览器(即使它们加在一起已经占有了当前浏览器市场 90% 左右的份额)中起作用的专用代码，那么我们可能会

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

毅然地决定使用 Java。

54.9 快速发展与相对稳定

JavaScript 是这两种语言中比较新的一种，所以，它也正在经历着更加频繁的一系列的更新与提高。在本书编写之时，Java 1.0.2 版已经相对稳定了，大多数的语句、操作符、语法等，已经有了很完善的定义，在后续的版本中很可能不会再有太大的变化。而 JavaScript 则正处于迅速发展中，从其原始目标为在客户端提供表单验证（而不是服务器端的 CGI），到具有仿真简单的游戏如 Pong 或突围的潜在能力等。JavaScript 所具有的特性包括：

- Java 与 JavaScript 之间的通信
- JavaScript 与插件之间的通信
- 确定已安装的插件
- 数据感染（Tainting）（安全性支持）
- 图像反射（reflection）（动态显示所有图像）
- 新事件处理器（onMouseOut 等）
- 对 < script > 标签的新属性（SRC）
- 新的内置对象（数组、字符串等）
- 新的操作符（typeof）
- 对象原型

正如我们可以看到的，JavaScript 正处于不断的发展变化之中，它为非程序员（或初级程序员）实现如下的任务提供了强有力的支持：

- 访问 Java 小程序的方法
- 允许插件和小程序与其它元素（其它插件、脚本及小程序）进行通信
- 客户端表单信息验证（在发送回 Web 服务器之前）
- 动态或基于环境变量（时间、日期、地点等）生成 HTML
- 完全使用 JavaScript 创建简单交互程序（例如 TicTacToe 游戏）

这种迅速的更新也有缺点，那就是风险。最初，Netscape 的每个版本所增加的新特性都被认为是一件好事。这在大部分情况下确实如此，但是现在，当我们坐在最新版本的浏览器面前，并开始使用 JavaScript 编程时，就会发现并非如此。我们肯定会自问，正在使用的特性是否在大多数的听众（特别是我们自己的站点的听众）那里能得到支持？并不是所有人都每隔几个月就更新其浏览器版本，所以不同版本将会把我们的听众分为不同的部分。另外一个缺点是，对增加到一种语言的每一个特性来说，也同时打开了引入程序陷阱的可能性。

就目前来说，一个好的策略是只使用那些自 Netscape 2.02 版以来就一直在使用的 JavaScript 特性。如果使用了 3.0 版特有的特性，一定要在页面上指出这一点，以通知我们的页面访问者。所以，Java 与 JavaScript 之间的另一个区别是，作为一种语言，Java 功能强大相对稳定，而 JavaScript 则正在随着不同的版本出现不断地长成壮大。

看到一个脚本编写者可以使用 JavaScript 完成如此多的工作，真是令人激动。在 3.0 中，笔者已经看到了简单的绘画程序、Pong 游戏以及每秒钟变化一次的 LED 时钟。在这种脚本中，大部分的内容将与它们的 Java 伙伴不再有明显的差别。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意：如果我们发现了新的特性，想添加到 JavaScript，那么唯一的机会是与 JavaScript 的开发者(NetScape 的 Brendan Eich)谈一谈。笔者希望 JavaScript 特性的扩充能够继续下去，特别是现在正与微软的 Visual Basic Script 竞争的时候。如果读者发现 JavaScript 在某个方面的限制太严，那么你可能会在未来的版本中对它进行修改。Brendan 将努力回应他所收到的所有电子邮件，但是由于数量太多，如果全部回应的话，可能要占用他的一些时间。

54.10　库

Sun 公司发行 Java 时带有一套标准的库，这大大提高了 Java 的实用性。程序员无需再自己编写用来处理图像、套接字等内容的代码，Sun 公司已经为他们实现了这些处理。我们只要学习标准的 API，就可以快速编写出终端模拟器、字处理程序等软件。

由于 JavaScript 本身还相对较为年轻，还没有时间可以建立起标准的代码汇总，用于基于 Web 的应用程序开发。阻碍这一进展的一个主要问题是，我们必须在 HTML 文档中嵌入想要使用的脚本代码。随着 <script> 标签中 SRC 属性的实现，现在可以在单独的文件中编写自己的代码了，在页面中只需引用所编写的脚本即可。这与 <applet> 标签中的 CODE 属性类似(在下面的例子中，一个页面将加载所有 JS 文件，在窗口的顶端显示正确的标题，并在 Welcome to my home page 文本上面显示出时钟)：

```
<html>
<head>
<script language = "JavaScript" src = "header.js"> </script>
</head>
<body>
<script language = "JavaStript"
src = "http://www.foo.com/scripts/timer.js"> </script>
<script language = "JavaScript" src = "body.js"> </script>
</body>
</html>
```

- HEADER.JS:

```
document.write("<tittle> Welcome! </tittle>");
alert("Welcome To My Homepage!");
```

- BODY.JS:

```
document.write("Welcome to my home page!");
```

通过 SRC 属性引用一个 JavaScript 文件的功能，使我们可以比以前更方便地重用脚本。可以预期，能够方便地访问的标准代码库的出现只是个时间问题。具有讽刺意味的是，虽然 Java 更为复杂，但我们目前可以找到的标准 Java 代码要比 JavaScript 多。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

54.11 JavaScript 和 Java 集成

关于 Java 与 JavaScript 之间的区别还有许多其它的例子，例如内存需求(及限制)、线程(Java 中有线程而 JavaScript 中没有)等等。但笔者认为，本章所例举的这些区别，或许会有助于加深我们对 JavaScript 的理解。

应该认识到，JavaScript 并不是 Java 的一部分，它是一种补充性的语言，可以使我们充分地控制浏览器的行为。我们现在可以将更多的计算和交互性工作从服务器移交给用户的客户浏览器——以减轻服务器的负担，提高服务器的性能。JavaScript 并不是一种全功能的或通用性的描述语言，但是在所限制的 HTML、插件、浏览器事件以及窗口这些范围内，作为向 Web 页增加交互性功能的简单方式，JavaScript 非常出色。我们将在下一章中看到这些。

究竟是使用 Java 还是 JavaScript，这不仅取决于我们的编程技术，也取决于与 Web 有关的任务范围。仔细分析所要完成的任务，看看文本滚动、图标闪烁或计算器等是否更容易由 JavaScript 来实现。如果我们需要使用特殊化的文本来控制浏览器窗口的大部分区域，或者有一个极其复杂的应用，那么当然应选择 Java。

下一章我们将介绍 JavaScript 的语法，提供对 JavaScript 能力更全面的介绍。我们可能会发现使用 JavaScript 编程与创建 Java 小程序一样有趣。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 55 章　JavaScript 入门

本章内容：

55.1　基础
55.2　第一个脚本
55.3　事件
55.4　使用事件句柄
55.5　变量
55.6　变量名
55.7　变量的作用范围
55.8　常量
55.9　表达式和操作符
55.10　控制语句
55.11　JavaScript 的函数
55.12　数组
55.13　内置函数
55.14　对象
55.15　一个实例

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

55.1 基础

到现在,我们已经学习了 Java 最新版本的大量内容,并且在前一章中,我们介绍了在 Web 开发中 Java 的伙伴——JavaScript。我们或许会感到奇怪(如果我们直接跳到这一章或正在书店阅读本书),为什么在一本关于 Java 的书中会有一章来介绍 JavaScript 呢? 原因很简单,JavaScript 在 Web 浏览器环境下补充了 Java。它允许那些有着很少或没有编程经验并且由于 Java 的复杂性而胆怯的人们,能够创建基于 Web 的交互式应用程序。

JavaScript 是一种松散地基于 Java 的描述性语言。通过在 HTML 文档中嵌入 JavaScript 代码,我们可以更有效地控制用户,并把大量的计算性工作(最初只有通过 CGI 脚本来获取)转交给客户端的浏览器。在加载一个页面时,浏览器将顺序读入这些脚本并可立即执行命令——即使在加载完成之前也可能会影响到页面。

由于 JavaScript 嵌入在 HTML 文档中,它可能是以嵌入在 < head > 或 < body > 元素中的完整脚本形式存在,也可能是由直接写入 HTML 代码中的事件处理器组成。

在程序清单 55.1 中,我们可以看出使用 < script > 标签在一个文档中建立 JavaScript 脚本框架的情况。

```
< SCRIPT LANGUAGE = "JavaScript" >
< ! - - HTML comment tags to hide script from old browsers
[JavaScript statements...]
// End hiding the code from old browsers - - >
< /SCRIPT >
```

从这个例子中可以看出,< script > 标签与嵌入 Java 代码时所使用的 < applet > 标签有些类似。SCRIPT 标签有一个称为 LANGUAGE 的属性,该属性允许我们指定浏览器解释随后的代码时所使用的语言。这使得 < script > 标签具有了通用性,我们最终可以使用这一属性嵌入 Visual Basic Script、TCL、Perl 等更多的脚本。

< script > 标签的另一个属性是 SRC。这一属性是 Netscape 3.0 实现的,它允许我们将所有脚本编写在另外一个文件中,然后引用该文件——而不需要把所有的语句都粘贴到 HTML 中。如果我们使用了 SRC 标签,则放置在 < script > ... < /script > 之间的内容都将被忽略。所以,对于不支持 JavaScript 的浏览器,我们可以放置另外一个 HTML。程序清单 55.2 使用了 SRC 属性。

```
< SCRIPT LANGUAGE = "JavaScript" SRC = "footer.js" >
You must not have a JavaScript Enabled Browser if
you see this (poor you!) Click < A HREF = "foo.html" > here < /A >
to go to another page.
```

Java学习群: 72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

```
</SCRIPT>
```

在程序清单 55.2 中，浏览器将加载包含在 FOOTER.JS 中的脚本，就像这些脚本已经输入到该 HTML 文档中一样。

55.2　第一个脚本

在学习 JavaScript 时，从基础开始非常重要。现在让我们立即启动浏览器（Netscape 的 Communicator 或微软的 Internet Explorer），并测试所学习的内容。

让我们从一个简单的脚本开始，并解释加载这个页面时将发生的事情。程序清单 55.3 是一个典型的“Hello World!”程序实例，该程序在测试一种新语言时非常流行。在这个例子中我们主要是通知浏览器显示字符串“Hello World!”，就像我们直接在 HTML 文档中输入一样。（注意，笔者通常对 HTML 或 JavaScript 标签使用大写方式，这只是一种编程习惯，这样做可使代码易于阅读。）

程序清单 55.3　使用 JavaScript 显示“Hello World!”

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!  --  Hide me from old browsers
document.write("Hello World!");
// End Hiding  --  >
</SCRIPT>
</HEAD>
<BODY>
Are you ready for JavaScript?
</BODY>
</HTML>
```

在浏览器中试运行这一程序，并观察结果。

基本来说，浏览器将这些代码读入 JavaScript 解释器，文本字符串“Hello World!”被传送给 document 对象的 write 函数，其结果使浏览器在一个新的页面上显示出短语“Hello World!”。注意，实际代码并不显示在浏览器窗口中。这是因为 HTML 分析器从不会接收到这些代码，当 HTML 分析器遇到 <SCRIPT> 标签后，它将被传送给 JavaScript 解释器。

55.3　事件

大部分情况下，我们创建脚本完成诸如存储信息、以一个特定表单显示数据、执行某些计算或响应用户动作（称为事件）等任务。JavaScript 所具有的所有元素使 JavaScript 成为了一

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

种功能强大的描述性语言，它可以处理所有这些任务。JavaScript 所完成的主要任务之一是捕捉并处理事件。我们对于浏览器所做的几乎所有响应都可以被 JavaScript 捕获。并且，我们的响应也可以触发其它事件或函数。

实际上，函数是在某些时间段上执行的存储代码块，或者当文档加载后立即执行，或者用于响应某些触发事件。可以将函数看作是指令集合，它可以使我们将某些需要一遍又一遍执行或重用的行为抽取出来。

当 JavaScript 遇到一个事件时，将其传送到事件句柄。事件句柄是指向将被执行的特定函数的标签。表 55.1 中列出了 JavaScript 中的事件及事件句柄。

表 55.1 JavaScript 中的事件及事件句柄

事件	事件句柄	触发事件的条件
blur	onBlur	在一个表单元素中，用户单击（或按 Tab 键）离开该元素时
click	onClick	在一个表单元素或链接中，用户单击该元素时
change	onChange	在一个表单文本、文本区域或选择对象中，用户改变值时
focus	onFocus	在一个表单元素中，用户单击（或按 Tab 键）到该元素时
load	onLoad	当页面加载后
mouseover	onMouseOver	当鼠标经过链接或锚点时
select	onSelect	在一个表单中，用户选择输入字段时
submit	onSubmit	在一个表单中，用户提交一个表单（单击 Sumbit（提交）按钮）时
unload	onUnload	用户离开页面时

55.4 使用事件句柄

虽然在 JavaScript 脚本的任何地方都可以使用事件句柄，但通常我们只在 HTML 表单元素或与锚点和链一起使用。这样做的原因是 JavaScript 使用 HTML 表单作为发送数据到我们的 JavaScript 脚本或对数据执行某些“预处理”的手段，而不仅仅是发送数据到服务器。

例如，我们站点有一个登记表单将寻问用户一些关于他们的问题，我们需要确认他们至少输入了他们的名字和年龄。在 JavaScript 之前，表单被直接提交回 Web 服务器，检查所填写的适当字段。如果没有填写，则表单被送回到用户，再次寻求适当信息。现在，使用 JavaScript 可以在发送表单之前就检查这些字段，请求用户填充信息，而不需要与远程服务器连接的种种开销。

关于如何在现有 HTML 代码中添加一个事件句柄，让我们来看一个例子。一般情况下，按照下面的语法进行：

```
< TAG eventHandler = "JavaScript code" >
```

当然，TAG 是某种 HTML 标签，eventHandler 是表 55.1 中所列出的事件句柄之一。“JavaScript code”可以是任何有效的 JavaScript 代码，但通常是该文档中早些时候已经装入的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

一个函数的调用。程序清单 55.4 说明了在公共超文本链接中一个嵌入的 JavaScript 事件句柄。当我们单击该链接时，一个对话框中显示出该文本，后面是一个 OK 按钮，单击该按钮，可以返回到页面。

程序清单 55.4　HREF 中的一个事件句柄

```
< A HREF = " # " onClick = "alert( 'Wow! It Works!' );" > Click here for a message!  < /A >
```

在本例中应注意以下几点：

- 在此处的 HREF 属性中没有 URL。为什么呢？当用户浏览弹出的窗口时我们可能并不需要浏览器转移到另外一个页面。当用户链接时，不仅 onClick 事件句柄被激活，而且浏览器还试图转移到 HREF 所指定的地址。在这个例子中，我们是在将这一链接用于 onClick 事件名柄，而不是其超链接引用。另外一种方法是输入：

```
< a href = "javascript:alert('Wow It Works!') > Click here < /a >
```

- onClick 是大小写混合的。虽然 HTML 不区分大小写，但 JavaScript 是区分大小写的，这是我们创建函数和变量时应牢记的重要的一点。
- alert(...)是在屏幕上产生一个警告对话框的标准函数。注意，该函数及 JavaScript 的所有函数与 Java 类似，都使用括号来包含它们的参数。在这个例子中，参数是字符串'Wow! It Works!'。另外还应注意所使用的引号是单引号。当我们需要在引号中再使用引号时，可以将单引号与双引号交替着嵌套使用。如果在一个元素中要使用多于两层的引号嵌套，我们可能需要另外想办法减少这种对引号的需求。
- 引号中的 JavaScript 代码"alert('Wow! It Works!');"以一个分号结束。在 JavaScript 中，我们使用分号结束一条语句，这与 Perl 及其它语言类似(包括 Java)。与 Perl 不同的是，JavaScript 中分号是可选的。

现在，我们已经看到了在 HTML 代码中实现 JavaScript 的两种主要方法(在 < SCRIPT > ... < /SCRIPT > 标签中包含脚本，或直接嵌入到 HTML 表单和链接)，下面再来看看 JavaScript 代码块的建立。

55.5　变量

要在 JavaScript 中创建一个变量只要使用关键字 var 声明它即可。在声明变量时可以使用某个值对变量进行初始化，但不是必须的。程序清单 55.5 给出了几个创建变量的例子。

程序清单 55.5　JavaScript 中的变量声明

```
var foo = 23
var a, b, c = "letter"
var aNumber = "99"
var isItTrue = false
var flag1 = false , bingo = null , star
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

与 Java 不同，JavaScript 可能是唯一一种不能显式地设置一个变量类型的语言，例如，将一个字符串强制转换为一个整型值。JavaScript 中也有类型，但它们是隐式设置的。这就是说一个变量的类型是由定义和使用它的上下文决定的。

当使用一个字符串值(如程序清单 55.5 中的 a、b 及 c)初始化一个变量时，则该变量为字符串类型；如果使用一个数值初始化一个变量，则该变量就是一个整型变量(如程序清单 55.5 中的变量 foo)。下面的代码在一个语句中放置了多个变量：

```
bax + bay + baz
```

这一代码试图以与第一个变量相同的类型来处理所有变量。如果 bax 是一个字符串，bay 和 baz 起初都是整型，那么 JavaScript 将把 bay 和 baz 作为字符串类型对待。JavaScript 变量的这种隐含类型的本质使得我们可以不考虑变量类型很方便地再次使用变量。

如果我们将某个变量 day 设置为“Tuesday”，而脚本又在稍后的部分将值 46 赋给该变量，则 JavaScript 解释器(在浏览器内部)并不报错。然而，正是因为这一点，我们在命名变量时必须小心，以避免变量作用范围相互覆盖，引发奇怪的错误。使用前面我们讲过的方法，在解释器窗口内声明和设置变量进行试验将会非常有帮助。(在 Netscape 中，只要在打开的 URL 窗口中输入 javascript:即可。)

表 55.2 中列出了 JavaScript 中可能的隐含数据类型，以及它们可能的值。

表 55.2 JavaScript 的数据类型

数据类型	值
数值	100、-99.99、0.000001
布尔	true(真)、false(假)
字符串	"this is a string"、"This is another"、"5555"
空值(null)	具有空值的一个特殊关键字

55.6 变量名

JavaScript 遵循与 Java 创建变量名时相同的命名规则。变量必须以一个字母或下划线作为开始，其后可以包含数字、字母或下划线。程序清单 55.6 中给出了 JavaScript 中可能的变量名样本。切记必须保证变量名的唯一性，并且 JavaScript 对大小写是敏感的。

程序清单 55.6 变量名样本

```
Too_hot
cold999
_100JustRight
This_is_a_long_variable_name_but_it_is_valid000
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

55.7　变量的作用范围

我们在前面提到过，必须保证变量名互不相同，以防止变量值相互覆盖，但是如果我们确实需要使用相同的变量名时会怎么样呢？这要涉及到变量的作用范围。全局变量可以在整个脚本及其函数中进行访问。局部变量则只能在创建它们的函数中访问。局部变量在函数结束时便消毁。要定义一个变量为全局变量只要给该变量赋值即可（例如“foo = 95”）。

在函数中定义一个局部变量，可使用 var 关键字。

55.8　常量

我们可以将常量看作是等号表达式右边的值。这是 JavaScript 中表示值的具体办法，与 Java 方法很类似。下面列出了各种常量及它们可能的值：

- 整数：

十进制表示为一个不以零开始的数字串：

(8988)

八进制表示为一个以零开始的数字串：

(077,056565565)

十六进制表示为以 0X 开始后跟以任何数字（包括：0 - 9 及 A - F 或 a - f）。

- 浮点数：

以一个零或更多的数字后跟一个句点“.”，再后面是一个或多个数字的形式表示。

以科学记法表示：一个数字串后跟 E 或 e，以及表示指数的一些数字（如 - 4.666E30）。

- 布尔值：

true（真）或 false（假）。

- 字符串：

由单引号或双引号引起的零个或多个字符。

字符串可以包含最终可以影响其显示方式的特殊字符：

- \ b——回退字符
- \ f——进行字符
- \ n——换行字符
- \ r——回车字符
- \ t——制表（Tab）符
- \ "——转义引号——这样就可以在一个字符串中显示一个双引号
- \ '——转义单引号

55.9　表达式和操作符

对于一个可以使用的语言仅仅有值是不够的，还需要对这些值进行有意义的操作。JavaScript 使用表达式来操作数字、字符串等。一个表达式是由常量、操作符、子表达式及带有值的变量组成的一个集合。我们可以使用表达式为一个变量赋值，如：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
today = "Friday"
```

也可以只计算表达式的值，如：

```
45 - 66
```

JavaScript 使用的表达式有：算术表达式，字符串表达式及逻辑表达式，它们的值分别是一个数值、一个字符串及 true 或 false。JavaScript 的操作符与 Java 非常相似，表 55.3 中汇总了 JavaScript 的各种操作符。

表 55.3 JavaScript 操作符

操作符	解释
计算类操作符	
+	数值相加和字符串连结
-	数值相减和单目取反
*	乘法
/	除法
%	取模(余数)
++	自增(前自增和后自增)
--	自减(前自减和后自减)
逻辑类操作符	
= =、! = =	等于和不等于(不是赋值)
<	小于
< =	小于等于
>	大于
= >	大于等于
!	逻辑非(NOT)
&&	逻辑与(AND)
||	逻辑或(OR)
?	三元条件选择
,	逻辑连接
位操作类操作符	
&	按位与(AND)
|	按位或(OR)
^	按位异或(XOR)
~	按位取反(NOT)
< <	左移位

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

操作符	解释
> >	右移位
> > >	无符号右移
赋值类操作符	
=	赋值
X=	自包含赋值(此处 X 可以是 + 、- 、* 、/ 、% 、& 、^ 、< < 、> > 、| 、> > > 、~),例如:A + = B 与 A = A + B 等价

操作符的优先级与 Java 相同。JavaScript 采用从左到右的简单求值模式,在计算一个表达式时,如果遇到其值为 false 的表达式,则不再计算表达式的其余部分并返回 false。如果要分组表达式,使某一部分优先求值,使用圆括号将其括起来。例如:

```
(56 * 99) + (99 - (44/5))
```

有一种便捷的表达式是条件表达式,这种表达式需要经常使用。这种表达式可以快速计算一些条件,返回两个值中的一个,其语法如下:

(条件)? 值 1:值 2

如果条件值为 true(真)则返回第一个值;否则,返回第二个值。例如:

```
isReal = (Imagination < = Reality)? true:false
```

55.10　控制语句

有了赋值操作符和数学操作符,就可以给变量赋值并进行简单的数学运算等操作了,但是仍不能编写出有意义的 JavaScript 代码。我们还需要有一些手段控制语句计算的流程、基于值进行选择、忽略某些语句以及对某些语句进行循环直到遇到某一条件等。

这就是控制语句的作用。JavaScript 将这些语句分为条件语句(if ...else)、循环语句(for、while、break、continue)、对象操作语句(for...in、new 、this、with)以及注释语句(//、/ * ... * /)。本节中将一一介绍这些语句并给出其相应的例子。(我们将在学习了 JavaScript 的对象模型后,再回过头来学习对象操作语句。)

JavaScript 使用大括号("{"和"}")括起一组语句,形成一个完整的语句块。当 JavaScript 遇到这种代码块时将同时处理完块中的所有语句(当然,除非 JavaScript 遇到了另一个分支,见稍后的介绍)。

55.10.1　条件语句

条件语句可以使我们的脚本根据所指定的条件做出选择。

if...else 语句　当需要根据某些条件执行代码块时,可以使用 if 语句。其语法格式为:

```
if(其值可能为真的表达式){
零个或多个语句 ...
}
```

如果想执行两个代码块中的一个,可使用 if...else 语句,该语句将导致执行一个块或另

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

一个块。其语法格式为：

```
if(其值可能为真的表达式){
某些语句 ...
}
else{
其它一些语句 ...
}
```

注意：如果语句块中只包含一条语句，可以省略大括号。然而，这不是我们所推荐的方式，因为这将使我们的代码在以后处理时很不方便。

程序清单 55.7 给出了实现 if...else 语句的例子。该程序还说明了将多个 else 与一个 if 语句结合在一起的方法。

```
if...else statement
if ( jobs < 100) && (money < = budget) {
poor = true;
free = (99 - x) / jobs ;
}
else if (jobs ! = overTime) {
workers = "Strike"
}
else {
poor = false;
workers = "Happy";
}
```

稍后，我们将介绍函数以及在 JavaScript 中构造函数的方法（参见本章的“JavaScript 的函数”一节）。现在我们只给出函数的一个实用定义即它是执行一个动作或返回一个值的一组指令。由于函数可以返回一个值，所以它也可以返回布尔值 true（真）或 false（假）。这样，我们就可以在 if 语句中作为测试条件使用函数调用。程序清单 55.8 给出了这样的例子。

```
if ( pagelsLoaded) {
alert ("All Done!");
done = true;
}
else {
done = false; }
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

55.10.2　循环语句

有时候我们需要反复执行一组语句直到满足某个条件。例如，让我们的页面反复播放一段背景音乐直到用户单击了“stop!”，或者让某个数反复被 6 除，直到小于 50，JavaScript 使用 for 或 while 结构执行这种操作。

for　for 循环重复执行某些语句直到满足某些条件。for 循环的语法结构实际上与 Java 的 for 语句完全相同。其语法结构如下：

```
for([初始表达式];[条件];[增量表达式]){
一些语句 ...
}
```

在建立 for 循环时要按照标准格式设置三个表达式。初始表达式可以任意复杂，但通常情况下只是一个计数器变量的初始化赋值。第二个表达式为条件表达式，每循环一次都要执行一遍该表达式，如果该表达式的值为 true（真），则再执行一次循环；如果该表达式的值为 false（假），则结束循环，解释器跳转到循环后面的第一条语句。增量表达式在每次循环时求一次值，通常是计数器变量增加 1 或减少 1。实际上也就是这样一个过程：初始化某些计数器、测试条件、如果条件为 true（真）执行括起来的一组语句、增加计数器、再次测试条件、等等。

> **提示：**虽然增量表达式不是必须的，但我们也应该使用它改变某些值，以便使条件表达式能够最终变为假。否则，循环将会永远执行下去（直到我们等得不耐烦了，重新启动计算机）。

程序清单 55.9 给出了一个 JavaScript 的 for 循环语句的简单例子。

```
<script language="JavaScript">
var myMessage = "Here we go again! <br>";
var numberOfRepeats = 100;
for ( i=0; i < numberOfRepeats ; i++ ) {
document.write(myMessage);
}
</script>
```

while　while 循环要比 for 循环稍简单一些，它每次测试一些表达式，如果这些表达式的值为 false(假)则退出循环。当所要测试的变量也要出现在循环所执行的语句块中时，我们可能会选择使用 while 循环。注意，在执行循环体语句之前，首先测试条件，并且对每一次循环，条件只测试一次。下面是 while 循环的标准语法格式：

```
while(条件表达式){
一些语句 ...
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
}
```

程序清单 55.10 中重复显示 tt 的当前值,直到 tt 大于等于 XX(此处 XX 为 55)。

程序清单 55.10 while 循环的一个例子

```
<script language="JavaScript">
tt = 0
xx = 55
while ( tt <= 55) {
tt += 1;
document.write ("The value of tt is" + tt +". <br> ");
}
</script
>
```

55.10.3 break 和 continue 语句

有些情况下,我们需要对 for 或 while 循环中的语句块做进一步控制。例如,有时候我们会强制跳出一个循环,并继续执行下一条语句;而有时候我们会要求停止执行当前循环而开始进入下一个循环。我们可以使用 break 和 continue 语句实现这些功能。break 语句使得 for 或 while 循环提前结束,使执行跳转到循环后面的第一条语句。continue 语句则停止当前一次循环并开始下一个新的循环。

提示:这些语句很容易混淆,它们的作用会随着时间的推移变得含糊不清。记住这些语句的功能的一个好办法是,想象 break 语句是把循环打破,使它再也不能执行,而程序本身将继续执行下去。想象 continue 语句是不管下面的语句是什么都跳过这些语句重新开始。程序清单 55.11 和 55.12 与程序清单 55.9 和 55.10 相对应,说明了这些控制语句的使用方法。

```
<script language="JavaScript">
var myMessage = "Here we go again! <br>";
var numberOfRepeats = 100;
for ( i=0; i < numberOfRepeats ; i++ ) {
document.write(myMessage);
if ( i <0) {
document.write("Invalid Number!");
break
}
}
</script>
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序清单 55.12　继续一个 while 循环

```
<script language = "JavaScript" >

var tt = 0;
xx = 55
while ( tt < = 55) {
tt + = 1;
if (tt < 0) {
continue;
}
document. write ("The value of tt is " + tt +". <br> ");
}
</script >
```

55.10.4　注释语句

每一种语言都需要有一些办法对正在进行的动作进行确切的说明，特别是当打算重用自己的代码时。常常是当我们全力以赴地编写自己的新代码时，一切看起来都是显而易见的，而若干天以后，我们会发现“我当时是怎么想的呢?”所以为自己的代码加上注释总是一个好办法。我们之所以把注释语句放在控制语句这一节来讨论，是因为注释语句实际上是在告诉 JavaScript 解释器这样一个事实：不管这段代码是什么内容都跳过这些代码。

注释语句类似于一个从初始化时就一直为 false(假)的 for 循环。JavaScript 支持两种注释：

- 逐行注释(//)
- 多行注释(/* ... */)

不管使用那一种注释，我们都可以在其中放置任何所需要的注释内容，只有一个例外。我们应该还记着关于使用 HTML 注释避免旧的浏览器错误地显示 JavaScript 代码的讨论吧。也就是说，我们不能在注释中使用 - - >，除非我们确实打算结束脚本。

还应注意，我们必须把单行注释放在 HTML 注释的结束符前面，因为 JavaScript 解释器不会把 - - > 作为具有什么含义的字符来识别，如果我们忘记在它前面使用//，解释器将给出错误信息。

那么，为什么放在开始的 HTML 注释行后面的初始行(例如“ Hide me from the old browsers”)不会给出 JavaScript 错误呢? 原因是解释器忽略了含有 < - - 的行的其它内容。这是很方便的，我们可以使用这一行来描述一下我们的脚本及其它一些内容。

程序清单 55.13 中给出了两种方法显示注释的例子。

程序清单 55.13　显示注释的一个例子

```
<html>
<script language = "JavaScript" >
<! - - Hide this code from old browsers
one = 1
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
two = 2
// three = 99 everything on this line is ignored....
four = 4;
five = 5; /* everything on this line, and all
subsequent lines will be ignored, until
we get to the closing comment */
six = 6;
// remember to comment out the last line if you are using the HTML comments also --->
//You must not have JavaScript if you see this line...
</script>
</html
>
```

55.11 JavaScript 的函数

我们现在将要介绍 JavaScript 最有意思的部分。大部分脚本的核心是由函数组成的。我们可以将函数看作是一个命名的语句组，它能够以参数形式接收其它的变量或语句。回忆一下 if 语句是如何构造的：

```
if(someTest){
零个或多个语句 ...
}
```

函数的构造也非常相似：

```
function someFunction(参数){
某些语句 ...
return someValue;
}
```

让我们来详细地讨论一下函数。正像我们在本章的前面提到的，函数是可以多次重用的代码块，只要通过名字调用该代码块并有选择地传送一些参数即可。函数构成了我们所要建立的大部分脚本的核心，它作为 JavaScript 的基础就像类对于 Java 一样。

JavaScript 带有许多内置的函数供我们使用，JavaScript 也允许我们自己建立函数。例如，假设我们要使用 JavaScript 建立一个小型 HTML 页，可以使用函数一一分离出所要完成的每一个子任务，这将使代码的修改、阅读及重用更加容易。请看程序清单 55.14。

程序清单 55.14 使用函数的一个简单例子

```
<html>
<head>
<script language="javaScript">
<! -- remember me?
var age = 0;
function myHeader (age) {
document.write("<TITLE>The" + age + "Year Old Page</TITLE>");
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
|
function myBody (date, color) |
document.write (" < body bgcolor = " + color + " >");
document.write ("< h3 > Welcome to My Homepage! </h3 >");
document.write ("The date is" + date + "< br >");
|
function manyLinks (index) {
if (index = = 1) |
return "http://www.yahoo.com";
}
else if (index = = 2)|
return "http://home.netscape.com";
}
else return "http://www.idsoftware.com" ;
|
// return the title
myHeader(33);
// done for the moment! - - >
</script >
</head >
< script language = JavaScript >
< ! - -
myBody("July 22, 1996", " # ffffff");
document.write(" < a href = " + manyLinks(2) + " > Here's a link! </a >");
// - - >
</script >
```

在这个例子中，每个函数都包括一些 HTML 代码。我们可以看到如何使用参数向每个函数传送信息，JavaScript 传送值时使用的是引用，即当向一个函数传送值时，实际上只是向函数传送了值的指针。（一个值的指针是一个地址，类似于信封上的房间地址所给出的如何找到这个房间的信息。）如果函数改变了所传送的值，则所改变的值对整个脚本有效，而不是只在该函数范围内有效。上述程序清单的运行结果如图 55.1 所示。

另外，注意其中的返回操作。我们可以有选择地向调用它的语句返回一个显式的值（如上例中的 return http://...）。返回值的类型可以是 JavaScript 的任何有效类型。如果没有值显式地返回，当成功地完成该函数时 JavaScript 返回 true（真）。

注意定义函数与调用函数之间的区别。定义函数（或存储到内存）时要使用到 function 关键字。使用函数名字在脚本中的其它地方调用该函数之前，函数中的任何一条语句都不会得到执行。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

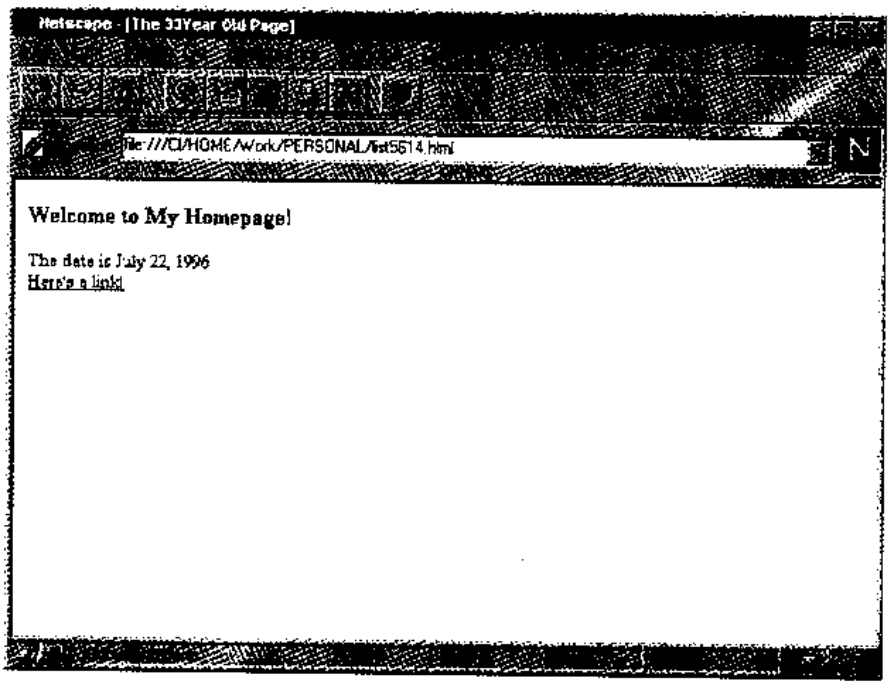


图 55.1 程序清单 55.14 的输出结果

警告:使用函数编写脚本时必须十分小心。JavaScript 读取脚本时自上而下,从左到右,对于尚未被读入的函数不能进行调用。

例如,假设我们有两个函数:myFirst( )和 mySecond( )。如果 myFirst( ) 出现在 mySecond( )之前,则 myFirst( )不能立即使用 mySecond( )函数。由于这种线性的代码解释和加载的原因,我们应当首先加载所有函数(通常在 <HEAD> 部分),然后再调用函数。所以将自己的函数放在 <HEAD> 元素中是一种好的习惯,这样可以保证我们的脚本开始执行命令之前加载所有代码。

记住,并不是必须向函数传送信息后才能使用它。我们也可以创建一个向页面写出许多 HTML 行的函数,构成 HTML 页的头部。一旦我们编写完这样一个函数,那么我们所需做的全部事情就是在需要时调用它——这样就减少了以后的大量输入操作。

55.12 数组

有了函数的概念,让我们再来介绍 JavaScript 另一个非常有用的结构——数组。数组是一些有序值的集合,可以通过一个公共的名字和索引(一个表示值在序列中位置的数字)来访问其中的每个值。在 Netscape 3.0 以前,我们必须通过使用在 Internet 脚本中常见的函数自己来创建数组。程序清单 55.15 给出了创建函数来构造一个数组的方法。

```
function MakeArray(n) {
this.length = n;
for (var i = 1; i <= n; i++) {
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
this[i] = "" }
return this
}
```

注意此处的新关键字 this。this 是一个特殊的关键字它指向当前的对象。稍后我们会讨论该关键字和另一个新关键字 new。为建立一个数组，我们只要为 MakeArray 的结果指定一个名字即可，如下所示：

```
Letterman = new MakeArray(10);
```

关键字 new 告诉 JavaScript，它右边的函数是一个对象构造器，JavaScript 将根据具体情况给出处理。为访问或设置新数组的值，使用如下语法：

```
Letterman[1] = "A list";
Letterman[3] = "Not so popular";
```

在 Netscape 3.0 中，数组是内置的，我们所要做的工作就是使用 Array 来代替 MakeArray 函数。这样上面的例子就变成了：

```
Letterman = new Array( );
```

设置数组的大小可以在初始化时指定，也可以通过为数组中最高元素赋予某种空值来设定。

55.13　内置函数

JavaScript 中也有一些内置函数，表 55.4 中列出了这些函数并为每一个函数给出一个简短的描述。

表 55.4　内置函数

函数	描述
escape(str)	将字符串转换为 HTML 特殊字符(如" "转换为%20)
unescape(str)	与 escape( )相反，将%20 变为" "
eval(str)	以 JavaScript 表达式对字符串 str 求值
parseFloat(str,radix)	将一个字符串转换为一个浮点数(如果可能的话)
parseInt(str)	将一个字符串转换为一个整数(如果可能的话)

我们已经了解了将语句组织在一起的函数，现在让我们再来看看对数据来说同样重要的与函数对等的另外一种结构——即对象。

55.14　对象

读者肯定已经阅读过本书的其它部分(除非一开始就跳到这一章)，所以一定遇到过有关

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java 中对象的介绍。基本上说，对象是组织数据以及与这些数据相关的操作的一种手段。在 Java 中，我们有类和方法的概念，但在 JavaScript 中，我们有对象和函数的概念。正如我们前面所提到的，JavaScript 带有许多非常有用的对象和函数。本节中我们将熟悉 Netscape 的对象模型并对它的每一个内置对象进行总结。

55.14.1 圆点指示符

JavaScript 借用了 Java 系统中访问属性和方法(JavaScript 不再区分函数与方法的说法)时使用的圆点指示符。

一般地，在访问信息时，应首先给出包含它的最高层对象的名字，然后是其它关于该信息的所有后续对象(或方法)。假设我们有一个对象称为 car，它包含一个对象称为 door，而 door 又包含另一个对象称为 doorhandle，doorhandle 包含一个方法称为 openDoor( )。我们可以以如下的语法形式随时使用该方法：

```
car.door.doorhandle.openDoor( )
```

假设 door 对象有一个属性称为 color，并且 color 有一个“Red”值。那么我们可以使用与如下方式类似的方法将该值赋给另一个变量：

```
myColor = car.door.color.value;
```

55.14.2 方法与属性

JavaScript 对象以属性和方法的形式组织数据。属性是与给定对象有关的被命名的值，属性访问通过对象来实现。在前面的 car 例子中，door 有一个属性 color。

属性是存储关于对象的信息方便且直观的方式。方法(或函数)是对对象的属性执行某些操作的一些代码块。另外，方法也可能将其结果存储在一个属性中。openDoor( )函数是 doorhandle 对象的一个方法。当我们在讨论 Navigator 内部建立的对象时，也将涉及到与之相关的方法及属性。

55.14.3 Window 对象

Window 对象是 JavaScript 的最高层对象，它包含了除 Navigator 对象之外的其它所有对象，Navigator 对象不与任何特殊窗口相关联。由于我们大部分的工作都是在 Navigator 窗口内完成的，所以这是一个我们应当熟悉的很有用的对象。

Window 对象中包含有打开和关闭窗口、产生警示框(只能单击 OK 按钮)、产生确认对话框(单击 Yes 和 No 按钮)、以及产生提示对话框(可以输入某些信息)等的方法。Window 对象中还包含有它所包含的所有框架的属性以及它所产生的所有子窗口。它还允许我们修改窗口底部的状态行(在这种地方我们可以看到许多页面的欢迎信息)。

表 55.5 中列出了 Window 对象的所有属性及方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 55.5　Window 对象的属性及方法

属性	描述
defaultStatus	状态行的缺省信息
document	窗口中当前所包含的文档
frames	描述窗口中所有框架(如果有的话)的一个数组
frame	一个框架(frame)对象
length	表示出窗口中的框架个数(如果有的话)
name	窗口的名字
parent	与窗口的名字同义。包含 frameset 标签
self	与窗口的名字同义,指向当前窗口
status	出现在窗口状态行的值,通常只持续片刻就被某些其它事件所覆盖
top	与窗口的名字同义,表示最顶层的窗口
window	与窗口的名字同义,指向当前窗口
location	指定当前文档的 URL 的一个字符串
方法	**描述**
alert	产生一个警告对话框
close	关闭窗口
confirm	产生一个使用 Yes 和 No 按钮及用户指定信息的对话框
open	打开一个新窗口
prompt	产生一个显示用户指定文本并提供一个输入框允许用户输入信息的对话框
setTimeout	以毫秒为单位为一个将要发生的事件设置时间
clearTimeout	重置以 setTimeout 所做的设置

55.14.4　Document 对象

Document 对象非常有用，它包含了大量关于当前文档的信息，并且可以使用 write 和 writeln 方法动态地创建 HTML。表 55.6 中列出了 document 对象的属性及方法,以及关于这些属性或方法的一个简短描述。

表 55.6　Document 对象的属性和方法

属性	描述
alinkColor	表示 ALINK 属性(在 < body > 标签中)
anchors	列出文档中所有 HTML 锚点的一个数组(< a name >)
anchor	一个锚点对象
bgColor	表示 BGCOLOR 属性的值
cookie	表示 Netscape cookie 属性的值
fgColor	TEXT 属性的值(在 < body > 标签中)
forms	列出文档中所有表单的一个数组
form	一个表单对象

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

属性	描述
history	包含当前浏览器历史的一个对象(已经访问过的链接、链接数、及链接 URL)
lastModified	文档上次修改的日期
linkColor	表示 <body> 标签的 LINK 属性
links	表示文档中所有 HTML 链接的一个数组(<a href>)
link	一个链接对象
location	文档的 URL
referrer	调用当前文档的那个文档的 URL
title	表示文档的标题
vlinkColor	表示 VLINK 属性中所列的颜色
方法	**描述**
clear	清除窗口的所有内容
close	在一个 open 调用之后,导致字符串缓冲区的内容写到屏幕上
open	开始将一个字符串写到屏幕上。需要一个 close 调用才能完成实际写操作
write	向当前窗口写一些表达式
writeln	与 write 相同,但在末尾处添加一个 newLine 字符

55.14.5 Form 对象

当 JavaScript 每次在 HTML 文档中遇到 <form> ... </form> 时便创建一个 Form 对象。该对象包含了所有存储在表单中的信息,可以用于向函数或服务器提交信息。表 55.7 中描述了 Form 对象的属性及方法。

表 55.7 Form 对象的属性及方法

属性	描述
action	表示 <form> 标签的 HTML 的 ACTION 属性
button	一个按钮对象(<input type = button>)
checkbox	一个复选框对象(<input type = checkbox>)
elements	一个列出表单中所有元素的数组
encoding	ENCTYPE 属性的值(用于 Netscape 中的 HTML 加载)
hidden	一个隐藏对象(<input type = hidden>)
length	表单中元素个数
method	<form> 的 METHOD 属性
password	口令字对象(<input type = password>)
radio	单选按钮对象(input type = radio>)
reset	重置按钮对象
select	选择对象(<select> ... </select>)

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

属性	描述
submit	提交按钮对象
target	<form>的 TARGET 属性
text	文本对象(<input type = text>)
textarea	文本区域对象<textarea>...</textarea>
方法	**描述**
submit	提交表单到 ACTION 属性中指定的位置

55.14.6　Navigator 对象

Navigator 对象与 Window 对象不同，它包含有适用于所有窗口的关于浏览器的信息。在 Navigator 3.0 中，JavaScript 增加了两个新属性——一个对象称为 mimeTypes，该对象列出浏览器可处理的所有 mimeTypes，另一个对象为 plug - ins，该对象列出浏览器可使用的所有注册插件。表 55.8 汇总了 Navigator 对象的所有属性(该对象没有相关的方法)。

表 55.8　Navigator 对象的属性

属性	描述
appCodeName	浏览器的代码名字，如"Mozilla"
appName	浏览器的名字，如"Netscape"
appVersion	包含浏览器的版本信息，如"2.0(Win95,I)"
userAgent	包含浏览器发送给服务器用来验证身份的用户代理头标，如"Mozilla/2.0(Win95,I)"
mimeTypes	一个表示所有可能的 MIME 类型的数组，这些 MIME 类型或者浏览器本身可以处理，或者浏览器可以将其传送到一个插件或一个帮助者应用程序(Netscape 3.0)
plug - ins	一个表示浏览器当前已经装入的注册插件的数组

55.14.7　String 对象

还有一些其它的 JavaScript 内置对象，这些对象既不是针对于浏览器的也不是针对于窗口的。这些对象中的第一个就是 String 对象。该对象非常有用，我们可以使用它的方法修改或添加 HTML 修饰，而不需改变字符串本身。关于该对象应当注意的一点是，我们可以将它的任意多个方法连接起来使用，创建多层次的 HTML 代码。例如：

```
"Hello!".bold( ).blink( )
```

该代码将返回：

```
<blink><b>Hello!</b></blink>
```

表 55.9 描述了 String 对象的属性和方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

表 55.9 String 对象的属性及方法

属性	描述
length	字符串中的字符个数
方法	描述
anchor	将一个字符串转换为一个 HTML 锚点
big	将字符串括在 <big> ... </big> 内
blink	将字符串括在 <blink> ... </blink> 内
bold	将字符串括在 ... 内
charAt	返回某个索引值位置的字符。索引从左至右,如果没有找到,则返回 -1
fixed	将字符串括在 <tt> ... </tt> 内
fontcolor	将字符串括在 ... 内
indexOf	在目标字符串中查找某一字符的第一个实例,并返回第一个字符的索引位置,如果未找到则返回 -1
italics	将字符串括在 ... 内
lastIndexOf	与 indexOf 相同,只是从右边开始搜索,找出搜索字符串中的最后一个实例,如果未找到则返回 -1
link	将字符串转换为一个超链接
small	将字符串括在 ... 内
strike	将字符串括在 <strike> ... </strike> 内
sub	将字符串括在 _{...} 内
substring	给定起始和结束索引,返回二者之间的字符串
sup	将字符串括在 ^{...} 内
toLowerCase	所有的大写字母都转换为小写字母(如 UpPeRcAsE 转换为 uppercase)
toUpperCase	所有的小写字母都转换为大写字母

55.14.8 Math 对象

Math 对象中既包含了一组允许我们对数值数据执行高级数学操作的方法,也包含了一组由常用的数学常量组成的属性。我们可以在脚本的任何地方使用 Math 对象,只要以如下所示的方式引用其方法即可:

```
Math.PI
```

我们也可以使用 with 关键字包含一系列的数学语句:

```
with(Math){
    foo = PI
    bar = sin(foo)
    baz = tan(bar/foo)
    }
```

表 55.10 列出了 Math 对象中的属性及方法。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

表 55.10　Math 对象的属性及方法

属性	方法
E	abs
LOG2E	acos
SQRT1_2	asin
LN2	atan
LOG10E	ceil
SQRT2	cos
LN10	exp
PI	floor
	log
	max
	min
	pow
	random
	round
	sin
	sqrt
	tan

55.14.9　Date 对象

我们要介绍的最后一个对象是 Date 对象。该对象允许我们抓取关于客户的当前时间、年、月、日等信息。另外，我们也可以快速地创建新日期对象，简化对事件之间日期或时间间隔的跟踪。我们甚至还可以对一个表示日期信息的文本字符串进行分析，使其作为一个 Date 对象在其它地方使用。该对象常常用来创建动态时钟，基于一天内的时间修改页面属性(如背景颜色)等等。表 55.11 为我们列出了 Date 对象的所有方法(它没有属性)。

表 55.11　Date 对象的方法

方法	描述
getDate	返回当前日期
getDay	从一个日期对象中返回星期
getHours	返回从午夜开始当前的小时数
getMinutes	返回某个小时过后的当前分钟数
getMonth	返回自一月开始的月数
getSeconds	返回当前时间的秒数
getTime	从指定的日期对象中返回当前时间
getTimeZoneOffset	以分为单位返回当前位置的偏移量(多于或少于 GMT 即格林威治时间的分钟数)
getYear	返回日期对象的年

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

方法	描述
parse	返回某个 Date 对象的当地时间自 1970 年 1 月 1 日 00:00:00 以来的毫秒数
setDate	以给定参数设置 Date 对象的日期
setHours	以给定参数设置 Date 对象的小时数
setMinutes	以给定参数设置 Date 对象的分钟数
setMonth	以给定参数设置 Date 对象的月份
setSeconds	以给定参数设置 Date 对象的秒数
setTime	以给定参数设置指定的 Date 对象的时间
setYear	以给定参数设置指定的 Date 对象的年
toGMTString	使用标准的 GMT 规范将日期转换成一个字符串(例如,Web, 24 Jul 12:49:08 GMT)
toLocaleString	使用当地标准而不是 GMT 标准将日期转换成一个字符串(7/24/96 10:52:02)
UTC	与 toGMTString 相反,转换一个字符串为自新纪元以来的毫秒数

55.15 一个实例

作为最后一个例子,我们用它来说明 JavaScript 到底能做些什么。程序清单 55.16 是每秒钟显示一次当前时间的一个 Web 页的源代码。我们可以在这个例子中看到本章前面所讨论的所有元素。实际上,该程序每秒钟获得一个 Date 对象;分析该对象的分钟、秒及小时数;将其转换成一个字符串,然后为该值设置一个表单输入字段。以这种方式使用一个表单在 JavaScript 中是常见的。文本输入字段不是用来输入数据,而是用来作为显示时间的一个"屏幕"。

程序清单 55.16 一个 JavaScript 时钟

```
<HTML>
<HEAD>
<TITLE>JavaScript Clock</TITLE>
<script Language="JavaScript">
<! -- Hide me from old browsers - hopefully
// Netscapes Clock - Start
// this code was taken from Netscapes JavaScript documentation at
// www.netscape.com on Jan.25.96
var timerID = null;
var timerRunning = false;
function stopclock (){
if(timerRunning)
clearTimeout(timerID);
timerRunning = false;
}
function startclock () {
// Make sure the clock is stopped
```

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
stopclock();
showtime();
}
function showtime () {
var now = new Date();
var hours = now.getHours();
var minutes = now.getMinutes();
var seconds = now.getSeconds()
var timeValue = "" + ((hours > 12) ? hours - 12 :hours)
timeValue + = ((minutes < 10) ?":0" : ":") + minutes
timeValue + = ((seconds < 10) ?":0" : ":") + seconds
timeValue + = (hours > = 12) ? " P.M." : " A.M."
document.clock.face.value = timeValue;
// you could replace the above with this
// and have a clock on the status bar:
// window.status = timeValue;
timerID = setTimeout("showtime()",1000);
timerRunning = true;
}
// Netscapes Clock - Stop
// end - - >
</script>
</HEAD>
<BODY bgcolor = " # ffffff" text = " # 000000" link = " # 0000ff"
alink = " # 008000" vlink = "800080" onLoad = "startclock()" >
<! - - main - - >
<table >
<tr>
<td colspan = 3 >
<form name = "clock" onSubmit = "0" >
<div align = right >
<input type = "text" name = "face" size = 12 value = " " >
</div>
<center> <b> <font size = - 1 > Welcome to My HomePage! </font> </b> </center> <p>
</table>
</BODY>
</HTML>
```

让我们来分析一下该程序，看它是如何创建出我们在浏览器下所看到的不断变化着的时钟的。

在初始的 HTML 代码启动页面之后，浏览器看到了 <SCRIPT> 标签，开始将代码传送到 JavaScript 解释器。HTML 注释 < - - 对老的浏览器隐藏起 JavaScript 代码。

接下来的三行是 JavaScript 将忽略的注释行。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

下面的两行将 timerID 初始化为 null(一个特殊的值,其作用是用来占据位置),timerRunning 初始化为 false(假)(一个布尔值)。变量 timerID 用在 setTimeout 和 clearTimeout 函数中。其作用只是作为跟踪特定向下计数器的名字。

接下来的五行定义了一个函数称为 stopclock,用来测试 timerRunning 的值是否为 true(真)。如果是,调用 clearTimeout,释放称为 timerID 的向下计数器。

下面的五行定义了一个称为 startclock 的函数。startclock 的全部工作是调用 stopclock 函数和 showtime 函数。在调用 showtime 之前停止时钟很重要,因为 showtime 重置了向下计时器 timerID。

再下来的十六行定义了该脚本的核心,称为 showtime。该函数创建了一个新 Data 对象称为 now;然后从该对象中获取小时、分钟及秒值;并将其分别赋值给变量 hours、minutes 以及 seconds。通过每次调用 showtime 时创建这一新对象,脚本就可以获得最近的时间,这就是为什么时间每秒钟都会变化的原因。

当小时、分钟及秒数从 Date 对象中取出之后,创建了一个新的变量 timeValue,该变量是一个 String 对象,并把校正后的小时值赋给该字符串(即 (hours > 12)? hours - 12:hours 表达式,它将 24 进制的小时数转换为 12 进制的小时数)。下面的赋值语句在 timeValue 字符串中添加了分钟值及秒数——并对 10 以下的数值进行了补 0 校正。

再下面的一行即:

```
document.clock.face.value = timeValue
```

将结果字符串放在了稍后所定义的表单文本输入字段中。通过将该值指定给这一字段,可使值显示在页面的这一框中。

函数 showtime 中的下面一行(三个注释行之后)启动了一个秒减计数器称为 timerID。一秒钟以后,再次调用 showtime 函数——实际上这是每秒钟都调用一次 this 函数的方式。

showtime 函数中的最后一行将 timerRunning 的值设为布尔值 true,它将影响到 stopclock 函数(中断由 timerID 所引起的一秒钟循环)。为了测试这一点,运行这一脚本,然后在 URL 输入窗口(浏览器窗口的顶部)输入:

javascript:stopclock( )

可以看到时钟停止了。在 URL 中输入:

javascript:startclock( )

则时钟再次运行起来。

在 showtime 函数之后,其余的行结束了脚本,并使用 HTML 创建了一个表,其中包含一个称为 clock 的表单,clock 表单中有一个称为 face 的输入字段。

注意在 < BODY > 标签中的 onLoad = "startclock( )"语句,当整个页面加载到窗口之后,触发了 onLoad 事件句柄,并开始调用 startclock( )函数,即启动了脚本。

Java学习群:72030155
好资料应该和你的朋友分享,把这本电子书分享给学Java的朋友,Java群空间;可以联系群主获取更多大型企业内部技术教程。

第十部分　Java 资源

第 56 章　Java 资源

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 56 章　Java 资源

本章内容：

56.1　Web 站点

56.2　新闻组

56.3　邮件清单

56.4　对移植问题的支持

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

56.1 Web 站点

跟踪像 Java 这样变化十分迅速的某种东西的最新信息，至少可以说是一件令人头疼的事情。所以，任何关于 Java 资源的清单在它完成之前就可能已经过时了。

我们并不准备提供所有涉及到 Java 的站点的庞大清单(Digital 的 Alta Vista Web 搜索引擎以"java"为关键字搜索时，可以返回成千上万条提示)，此处的清单只打算提供一些可以扩充的 Java Web 站点信息和一些可以寻找更多 Web 站点的好地方。所以，这些站点并不意味着是可以找到的全部站点——而只是我们开始建立 Java 参考书签清单的一个起始点。它们没有特定的顺序，也没有其它的衡量标准，只是笔者认为各种层次的 Java 程序员都会发现这些站点在某些方面会有帮助而已。

由于 Java 的主要特点之一是它能够把小程序嵌入到 Web 页面中，所以对于 Java 开发来说，Web 很自然地成为了一个很出色的信息源。这些站点提供了大量的信息，从 API 调用，到 Java 领域的最新消息——这对各种水平的 Java 程序员都是非常宝贵的资源。

56.1.1 Earthweb 的 developer.com

http://www.gamelan.com/

该站点原来的名字为 Gamelan(发音为"gamma - lahn")，它是所有 Java 资源站点的老祖先，提供了一个在 Web 上可以获得有关 Java 信息的庞大清单，从其广泛的 Java 小程序集到其它外部 Java 资源的列表，它是开始浏览查看其它 Java 程序员情况的一个很棒的站点。

56.1.2 Focus on Java(Java 焦点)

http://java.miningco.com

在 Focus on Java 的 Java Guid 中，John Zokowski 提供了一大批有关 Java 的特殊的信息。John 手工采集了最好的 Java 珍品，标识了最好的书和工具，并且提供了有关 Java 未来不同走向的见解。他的资源集是一个根据不同类型分类的 Java 资源库。

56.1.3 Inside Java

http://www.inside-java.com

Inside Java 是 Java 程序员一个很好的信息源。在这里，我们可以找到论文清单、Java 领域的最新情况、资源、Java 论坛以及 Java 闲聊室。这些东西可以使我们了解最新的变化、更新及版本等新潮流——是一个可以添加到我们的书签中的好 URL。

56.1.4 Java Applet Rating Service(JARS)

http://www.jars.com/

JARS 主要的焦点是提供 World Wide Web 上可以获取的有关 Java 小程序的排名。每一个小程序都要经过一个独立评判人员小组的检查，其中包括本书的一些作者，根据一套判定条件作出排名。如果一个小程序获得了排名所使用的指定总分，可以按照下列 JARS 奖励标准来区分它们：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- Web Applet 前百分之一
- Web Applet 前面分之五
- Web Applet 前百分之二十五
- Web Applet 前十名(当月)
- Web Applet 前 100 名(当月)

另外,对于提供源代码的小程序将进一步得到确认,并且如果可能的话提供一个到该源代码的链接。

如果想查看其它程序员的小程序并把自己的小程序与他人的程序进行比较,那么 JARS 是一个很好的站点。

56.1.5　Java Boutique

http://javaboutique.internet.com/

Java Boutique 有许多 URL 列表,包括小程序、评论、论坛、基本知识以及许多其它有趣的特性。

56.1.6　Java Developer's Journal(Java 开发人员杂志)

http://www.javadevelopersjournal.com/java/

Java Developer's Journal 中有免费 Java 讲座、免费试订阅三个月它的 Java Developer's Journal 杂志,免费软件等。该站点的特点是产品评论,如果我们想购买软件,则它可以给予我们帮助。

56.1.7　Java Developer Connection(Java 开发者连接)

http://java.sun.com/jdc

该站点是 Sun 公司的免费站点,提供最新的资料和预发行的软件——对每一个认真的开发人员是一个必须访问的站点。

56.1.8　Java Lobby

http://www.javalobby.org

Java Lobby 是一个由 Java 开发人员组成的群体,这些开发者致力于实现"一旦编写,随处运行"的 Java 承诺。他们走到一起为了保证 Java 永远保持纯洁。

56.1.9　Java Resources from Netscape(来自 Netscape 的 Java 资源)

http://developer.netscape.com/library/documentation/javalist.html

Java Resources from Netscape 提供了关于与其产品一起使用的 Java 技术信息的一个庞大列表。在这个站点,我们可以找到多个列表及第三方列表,它含盖了所有我们想知道的关于 Java 的内容。

56.1.10　Java World

http://www.javaworld.com/

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这是一个在线月刊。Java World 是 IDG 公司面向 Java 族的杂志。我们可以在这里找到链接到资源的信息以及有关基本知识。我们可以搜索指定的信息，并检查其“Nuts&Bolt's”部分，以获取关于软件用法方面的大量信息。

56.1.11 JavaBeans 站点

http://java.sun.com/beans

所有 JavaBeans 信息的根源就是这个出色的站点，它包括了有关信息及大量关于 JavaBeans 和 JavaBeans 项目信息的链接。

56.1.12 Sun 公司的主页

http://java.sun.com/

这可能是开始查找 Java 资源的最好地方。该站点是 Sun 公司的主页。在这里我们可以找到大量关于 Java API、JDK 以及 Java 语言本身的文档。我们可以下载最新版本的 JDK 及其它与 Java 有关的工具。任何认真使用 Java 编程的人员都应该经常全面地访问这一站点，我们还可以在这里找到其它地方不可能找到的即时性信息。

56.1.13 Javology —— Java 新闻及观点在线电子杂志

http://www.javology.com/javology

Javology 是一个在线杂志，其内容含盖了 Java 领域内当前所发生的事件。它的文章包括最近的新闻、对 Java 族中激进分子(mover and shaker)的采访、以及其它关于 Java 所发生事情的最新信息。Javology 可以帮助那些对 Java 感兴趣的人们了解 Java 最新的进展。

56.1.14 微软的 Java 主页

http://www.microsoft.com/java

微软的 Java 站点包括有关微软的 Java 实现方面的信息，并提供了到微软的 Java 工具的链接。注意，微软已经在修改其 Java VM 及 API 内核，如果需要了解这方面情况，可跳转到该站点。

56.1.15 Swing Connection

http://java.sun.com/products/jfc/swingdoc-current/

该站点是 Swing(JFC)项目的主页，提供即时提示和关于 JFC 使用的技术，以及关于即将发布的新特性的信息。

56.1.16 TeamJava

http://www.teamjava.com/

Team Java 的目的是通过提供关于工作机会、新闻、教育材料及其它有用的 Java 资源，向 Java 咨询者提供帮助。Team Java 还有一个“今天的小程序”服务称为 Java the Hut。总之，该站点对于使用或准备使用 Java 作为其专业环境的人们非常有用。即使是“周末 Java 勇士”也

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

将会发现该站点的价值。

56.2　新闻组

UseNet 新闻组也是一个巨大的信息源。但当人们不再给予帮助，开始争论他们喜欢争论的问题时，这些地方也可能成为一个主要的烦恼源。所以，如果我们熟悉 UseNet 并且在使用它时心情还不错，这些新闻组也是有价值的财富。如果我们并不熟悉 UseNet 新闻，最好先观察一会，对该系统有所了解，并远离火药味很浓的论战，在发出信息之前要阅读 FAQ 的内容。

在这样的前提下，值得一提的关于 Java 的 UseNet 新闻组有如下一些：

- comp.lang.java　Java 语言和编译
- comp.lang.java.advocacy　Java 支持者的讲话
- comp.lang.java.announce　Java 产品及其它服务公报(限制的)
- comp.lang.java.beans　JavaBeans 讨论与编程
- comp.lang.java.databases　Java 数据库编程
- comp.lang.java.gui　图像界面提示及帮助
- comp.lang.java.help　关于 Java 语言及编程的一般性帮助
- comp.lang.java.machine　Java 虚拟机讨论
- comp.lang.java.programmer　Java 程序员帮助
- comp.lang.java.security　Java 安全性讨论
- comp.lang.java.softwaretools　Java 工具讨论，帮助我们提高效率
- alt.www.hotjava　HotJava World Wide Web 浏览器

注意：要知道并不是所有新闻服务器都向它们的订阅者开放新闻组的 alt.层。如果无法定位这种地址，与新闻管理员联系。

Syracuse 大学的东北并行体系结构服务器(Northeast Parallel Architecture Server)跟踪 comp.lang.java 及其它一些新闻组。这是一个获取所有 comp. lang.java 发布的有关信息如垃圾收集的方便途径。

56.3　邮件清单

除了由 Java - SIG 管理及由各种小型组运行的邮件清单外，还有一些运行于 Sun 公司之外的邮件清单。

下面是清单的地址：

java - interest@java.sun.com

注意：这是一个客流量很高的组，每天有超过 20,000 位的订阅者及几十封的发布信息。该清单是不加限制的，所以如果我们很容易被压垮的话，这里不是一个适合的去处。

通过向如下地址发送信息，其正文中包括 subscribe java - interest 文字，我们可以订阅该

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

清单：

majordomo@java.sun.com

所有关于 Sun 清单的传输都被发到 comp.lang.java，所以不必既阅读邮件清单又阅读新闻组。关于 Sun 公司的邮件清单更详细的信息，可参考如下地址的内容：

http://java.sun.com/mail.html

56.4 对移植问题的支持

Java 是一个很流行的语言，有许多人都在努力地工作，将其移植到尽可能多的平台上，以期待它真正变成一种通用性的语言。下面的清单可以告诉我们与移植者们连接的一些地址。

56.4.1 Amiga 移植问题

瑞典的 Mattias Johansson(matj@o.lst.se)运行了到 Amiga 或 P'Jami 的移植 Java。

下面是三个电子邮箱地址：

amiga - hotjava - dev@mail.iMNet.de

这是一个封闭清单。参与者必须得到清单管理员的许可才能进入。

amiga - hotjava@mail.iMNet.de

这是一个开放的邮件清单，用于交换信息。为了订阅这一清单，向如下地址发送包含 subscribe amiga - hotjava 的信息：

mafordom@mail.iMNet.deamiga - hotjava - announce@mail.iMNet.de

最后的清单用来广播 Amiga 移植的声明。要订阅它，向地址 majordomo@mail.iMNet.de 发送包含 subscribe amiga - hotjava - announce 文字的信息即可。

56.4.2 DEC Alpha OSF/1 移植

这一部分包含了关于补订及 DEC Alpha 移值的信息。Web 页面由 Greg Stiehl 维护。

Web 站点：http://www.NetJunkies.Com/Java/osf1 - port.html

电子邮箱地址：stiehl@NetJunkies.com

56.4.3 Linux 移植问题

Linux 是免费的、IBM 兼容的 UNIX 版本。Karl Asha(karl@blackdown.com)为有兴趣在 Linux 下移植和使用 Java 及 HotJava 的人们维护着若干资源。

其 Web 站点地址为：

http://www.blackdown.org/java - linux.html

对于 Linux 问题有两个邮件清单：java - linux 和 java - linux - announce。第一个是一个讨论清单，第二个是一个广播清单。

邮件清单的地址为：java - linux@java.blackdown.org

要订阅这一清单，向下述地址发送以 subscribe 字样作为主体行的一个信件即可：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

java - linux - request@java.blackdown.org

或：

java - linux - announce - request@java.blackdown.org

Linux Java 移植的一个匿名 FTP 发布地址为：

ftp://substance.blackdown.org:/pub/Java

56.4.4　NEXTSTEP 移植问题

Bill Bumgarner(bbum@friday.com)维护着一个开放的邮件清单，其主题是关于针对于 NeXT 平台的移植及集成奥秘的讨论。

为了订阅这一清单，向下述地址发送一包含 subscribe 字样的邮件即可：

next - java - request@friday.com

该清单的电子邮箱地址为：

next - java - @friday.com

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群

72030155

进群可以免费获取视频教程以及每日免费听老师讲课

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。